REPORT DOCUMENTATION PAGE

. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
. DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 53	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
. ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

1. TITLE (*Include Security Classification*)
INFORMATION ENGINEERING AND THE INFORMATION ENGINEERING FACILITY VERSUS RAPID APPLICATION DEVELOPMENT AND FOCUS (UNCLASSIFIED)

2. PERSONAL AUTHOR(S)
Clark, Lucille C.

3a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Master's thesis	FROM _____ TO _____	December 1992	231

6. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. government.

7. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Computer-Aided Software Engineering, CASE, Information Engineering, IEF, Information Engineering Facility, FOCUS, Rapid Application Development, Methodology

9. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The Management Information Systems Department of the Naval Postgraduate School (NPS) is considering using the information engineering methodology with Texas Instrument's Information Engineering Facility (IEF), an integrated computer-aided software engineering toolset, for application development. The costs and benefits of introducing information engineering and IEF versus the rapid application development methodology and fourth generation language, FOCUS, were analyzed through a case study developed in both IEF and FOCUS. IEF offers a one model implementation, a standard computerized methodology, consistency checking, management tools for the application developer, and superior diagramming features and screen design whereas FOCUS offers rapid prototyping, numeric functions, a report facility, security within the data model, inherent database management facilities and excellent documentation. The benefits of IEF did not outweigh its costs. RAD and FOCUS were determined to be the methodology and tool of choice respectively for application development for the MIS department.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Barry A. Frew	22b. TELEPHONE (*Include Area Code*) (408) 646-2392 / 22c. OFFICE SYMBOL 05

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Information Engineering and the Information Engineering Facility verus Rapid Application Development and FOCUS

by

Lucille Charlotte Clark
B.A., Princeton University, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
December 1992

ABSTRACT

The Management Information Systems Department of the Naval Postgraduate School (NPS) is considering using the information engineering methodology with Texas Instrument's Information Engineering Facility (IEF), an integrated computer-aided software engineering toolset, for application development. The costs and benefits of introducing information engineering and IEF versus the rapid application development methodology and fourth generation programming language, FOCUS, were analyzed through a case study developed in both IEF and FOCUS. IEF offers a one model implementation, a standard computerized methodology, consistency checking, management tools for the application developer, and superior diagramming features and screen design whereas FOCUS offers rapid prototyping, numeric functions, a report facility, security within the data model, inherent database management facilites and excellent documentation. The benefits of IEF did not outweigh its costs. RAD and FOCUS were determined to be the methodology and tool of choice respectively for application development for the MIS department.

TABLE OF CONTENTS

I. INTRODUCTION

A. PROBLEM DESCRIPTION

The Dean of Computer and Information Systems at the Naval Postgraduate School (NPS) is considering using the information engineering methodology with Texas Instrument's (TI) Information Engineering Facility (IEF), an integrated computer-aided software engineering (CASE) toolset for application development. Currently, the programming staff uses a rapid application development (RAD) methodology and the fourth generation language FOCUS. The purpose of this thesis is to determine, through a representative case study, the costs and benefits of introducing information engineering (IE) and IEF versus using the current rapid application development methodology and tool, FOCUS. The results will determine, in part, which methodology and tool will be used to develop future projects of the Management Information Systems (MIS) Department.

B. RESEARCH QUESTIONS

This thesis does not attempt to devise a radical new approach to application development but rather evaluates how certain methodologies and tools can contribute to the MIS department. It presents an overview of the software development life cycle, the software development environment,

and several software development methodologies; an overview of fourth generation languages and computer-aided software engineering (CASE); a specific overview of FOCUS and IEF and the software development environment of the MIS department; and an evaluation of the tools and their associated methodologies based on a case study.

The case study consists of a bounded business area of a middle-sized enterprise. A business area is considered to be sufficiently bounded and constrained when (1) the accessed data (2) the processes including their timing and coordination (3) the business relationships with all their intricacies and (4) the business rules and policies affected by the processes and flows are all well known and clearly defined. (Haas, 1991) The business area for the case study is the Minor Property Accountability System and the medium-sized enterprise, the Naval Postgraduate School. The direct and indirect research questions to be answered are the following:

1. Is it worth the cost and effort to introduce information engineering and CASE as the methodology and tool respectively for a bounded business area for a medium-sized enterprise that uses rapid application development and fourth generation languages?
2. How does the information engineering methodology with IEF compare to rapid application development with FOCUS in terms of its costs and benefits?
3. What is the learning curve associated with the Information Engineering Facility (IEF)?
4. What are some of the effective techniques for evaluating methodologies, tools, and the software development environment?

One should not generalize the results of this thesis to every software development environment, organization, methodology, fourth generation language or CASE tool. Nevertheless, the results could be used as a guide to determine the costs and benefits of introducing information engineering and/or CASE to a rapid application environment which uses fourth generation languages for a medium-sized organization. Other organizations may use the interpretation of the analysis and results to fit their requirements and environment.

C. INVESTIGATIVE METHODOLOGY

Initially, a literature review of the general topics of software methodologies, fourth generation languages, information engineering, and CASE was undertaken. Specific research into the publications and vendor literature of FOCUS and IEF followed. The Minor Property Accountability System was chosen as the case study because it was developed using both methodologies and tools and was of limited scope and complexity. It was first developed by a MIS application developer with RAD and FOCUS and then separately by the author with IE and IEF. To determine the costs and benefits of the current system, extensive interviews with the MIS Director, the application developers, vendor, and other users were conducted.

The methodologies and tools were analyzed according to several subjective evaluation criteria and investigative approaches. "Hands-on" experience and training in business area analysis and business system design with IEF was used to determine the costs and benefits of using information engineering and IEF. The current Minor Property Accountabilty System developed in FOCUS was used to compare IEF with FOCUS. Note that a comprehensive evaluation of FOCUS or IEF to other fourth generation languages or CASE tools was not conducted.[1]

D. STRUCTURE OF THE THESIS

Chapter II provides the definition and classifications of a software development methodology and the software development environment; an overview of the software development life cycle and its criticisms; an overview of several software methodologies (prototyping, information engineering, and rapid application development) including their advantages and disadvantages; and a discussion of a tailored and universal approach to software development.

Chapter III provides a discussion of the relationship of a software tool to its methodology; an overview of the general characteristics of fourth generation languages and CASE tools;

---

[1]For a survey report on assessing the strengths and weaknesses of development tools in the CASE, 4GL and DBMS mainframe arena, the reader can order Computing Future's Benchmark Series Survey Report, Chilington House, East Chiltington, Lewes, East Susses, U.K. FAX 44 (273) 890375 Tel 44 (273) 890097.

a specific overview of FOCUS and IEF respectively; and concludes with a brief summary of the interrelationships of the methodologies and software tools.

Chapter IV provides the background information for the case study, the Minor Property Accountability System. It describes the application development environment of the MIS department to include MIS' software development methodology and the organization it serves. The functions of the Property Management Department and the system requirements of the Minor Property Accountability System are also documented.

Chapter V provides the analysis of the two methodologies and tools based on the case study. It describes the problems involved in evaluating methodologies and tools, the investigative approach used, the implementation of each facility in each tool, the evaluation, and a summarized conclusion.

II. SOFTWARE DEVELOPMENT METHODOLOGIES AND THE LIFE CYCLE

The software methodologies and tools presented in the following two chapters represent, for the most part, the views of their proponents. Therefore, the reader should be skeptical of any claims, especially by the vendor or source, and realize that some of the literature serves as marketing material. For example, James Martin's books are obviously slanted toward the CASE tool, Information Engineering Facility (IEF). In many cases, a new method is simply disguised as old practices with new terminology. When conflicting opinions have been published, they have been included. Unfortunately, most of the literature is disappointedly "party-line."

Notwithstanding, the purpose of the second and third chapters is to present the background material for the evaluation by explaining the concepts, advantages, and disadvantages of software methodologies and tools. The case study itself will compare two specific methodologies, RAD and IE, and two specific tools, FOCUS and IEF, and provide the basis for a comprehensive analysis and evaluation.

A. DEFINITION AND CLASSIFICATIONS

To avoid software development that is haphazard, unplanned, and unstructured, a software development methodology is followed. In general, a methodology is a

systematic approach to solving a problem by prescribing a set of steps and deliverables as well as rules to guide the progress and analysis of work. Specifically, methodologies include (1) the sequence of tasks (2) the outputs and the deliverables from each task (3) a description of how to perform the task and the personnel and training required, if applicable, and (4) guidelines on how to succeed and what pitfalls to avoid (Martin, 1991). Other characteristics can include that a methodology be recorded, teachable, measurable, and even automated.

With respect to software development methodologies, they are based on an understanding of system development, how it should be modelled, what the relevant design tools are, and how they should be supported (Floyd, 1986, p.31). Software development methodologies must also satisfy management's requirements of minimizing project risk, minimizing cost, ensuring timely delivery, and ensuring optimal use of project resources. From a business perspective, the loss of assets, the loss of customers, and not to mention, the loss of revenue could result from poorly designed systems.

Many organizations confuse the techniques used by a tool with the methodology. As Uluakar (1991, p.2) states, the techniques or procedures used to implement the methodology may vary with the available technology (such as CASE tools) and with experience. Tools simply automate the tasks and techniques, although some tools may impose a certain

methodology. For successful implementation, a software methodology must "mesh in concrete features and in the abstract with the application domain, the organizational approach to software development, and the organizational environment." (Ginsberg, 1988, p. 19-9)

Current software development methodologies fall into three general categories as illustrated in Figure 2.1: project management methodologies which are more concerned with management issues rather than the execution of individual project phases; methods and techniques methodologies which focus on the execution of the development cycle and tend to address only one or a combination of phases; and integrated methodologies which cover all phases of development, although precedence is usually given to the methods and techniques than to project management. Integration problems can occur when combining several method and technique methodologies in order to cover all phases of the software life cycle. Integration is usually supported by an automated tool such as CASE. (Jaakkola, 1991)

Whitten (1989, pp. 110-129) classify methodologies according to whether they employ process modeling such as structured programming and systems analysis and design; data modeling such as information engineering and object oriented design; or working modelling such as prototyping. However new methodologies blend process, data, and working models.

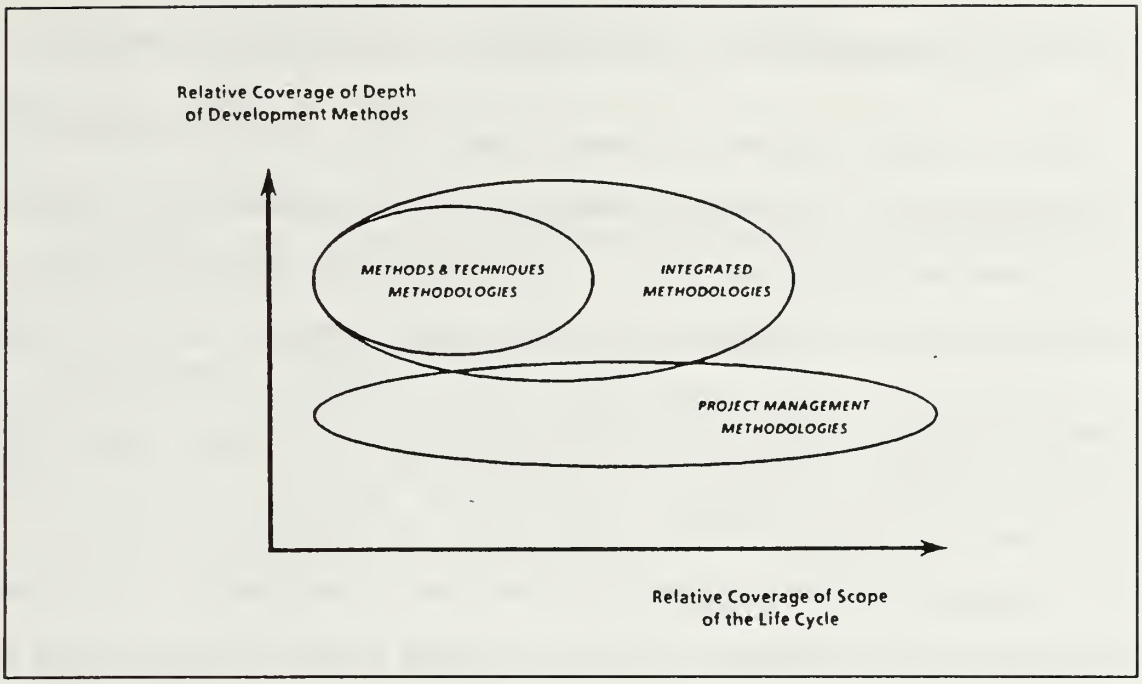


Figure 2.1 Scope and Depth of Software Development Methodologies (Jaakkola, 1991, p. 7)

B. METHODOLOGY CHARACTERISTICS AND EVALUATION QUESTIONS

Some characteristics of methodologies, also used as evaluation criteria, are as follows (Ginsberg, 1988, p. 19-8):

- the activities covered by the method
- the extensiveness of the method
- the method's appropriateness for various application areas
- the method's ability to incorporate the requirements of the target system
- the method's support of user involvement
- the method's ability to incorporate change
- the method's support of project management
- the availability of automated tools supporting the method

- the availability of training to support the method

Floyd (1986) presents some additional characteristics of methods such as their relation to a theoretical basis of systems development, be it a structured theory of analysis and design or the concept of systems development as a process of communication and cooperation; and the coherence of a method, how closely related are its guidelines and whether the method is based on **one** overall strategy.

Martin (1991) presents what he considers good and bad properties of methodologies. A good methodology should:

- be fully adaptable to circumstances versus being rigid
- minimize manual work versus being work intensive
- assume developers are intelligent and creative versus a "bureaucratic" approach in which developers are not allowed to think on their own
- computerized so that the methodology can be easily adapted and integrated with expert and project management systems versus paper methodologies
- provide proven guidelines for success, warnings of the pitfalls involved, and checklists so that the developer can apply these guidelines intelligently and flexibly versus an inflexible set of tasks
- make sense to those who use the methodology versus the developers not knowing why certain tasks are really necessary

A successful methodology should act as a guide to development (not a burden) allowing the developers trained in the tools and techniques to use their own initiative and creativity to build a quality system to meet their users' needs.

Other key questions for evaluating a methodology include (Ginsberg, 1988, p. 19-9):

- Does the method provide an effective means for developing, analyzing and communicating the project requirements and the resulting design?
- Does the method mesh with the existing organizational development style?
- Do the benefits of technology justify the training time?
- Is there available information about the method's use on comparable projects?
- How did the use of the method/tool affect those projects?
- What other factors affected the success/failure of those projects?
- Is the methodology clearly documented? Does it focus on deliverables instead of activities?
- Can the developer use metrics with it?
- Is it CASE tool independent?
- Does it cover the entire life cycle including maintenance?

Floyd (1986, p. 31) states "that you have to place yourself within the system development process as viewed by the method to really understand it." Indeed, that is the objective of the case study of this thesis.

C. THE SOFTWARE DEVELOPMENT ENVIRONMENT

Of special consideration when selecting a methodology is the software development environment (SDE). How does the methodology fit the organizational environment and the application area(s)? The SDE consists of all the resources

necessary to engineer software: the methodology, the tools, and the people (customer, developer, user, maintainer, management, etc.). Figure 2.2 illustrates the major elements of the SDE. According to Corbin (1991, p. 28) "Some call this [the SDE] a software management process. Others call it idealistic." The overall objective of the SDE is to reduce system development costs, maintenance costs, and personnel turnover.

Specifically, the benefits include:

- improved problem definition
- selection of the right problem according to the customer
- joint customer/IS responsibility and accountability
- acknowledgement of customer ownership of the system
- reusability of software, models, and data definitions
- acceptance of a consistent methodology
- productivity improvements through teamwork and development support tools

A systems development methodology is just one part of the SDE and provides consistency from one project to another, reducing training. Other benefits include improved system quality, reduced development and maintenance costs, increased team productivity and a reduction in the time to implement business strategies. (Corbin, 1991)

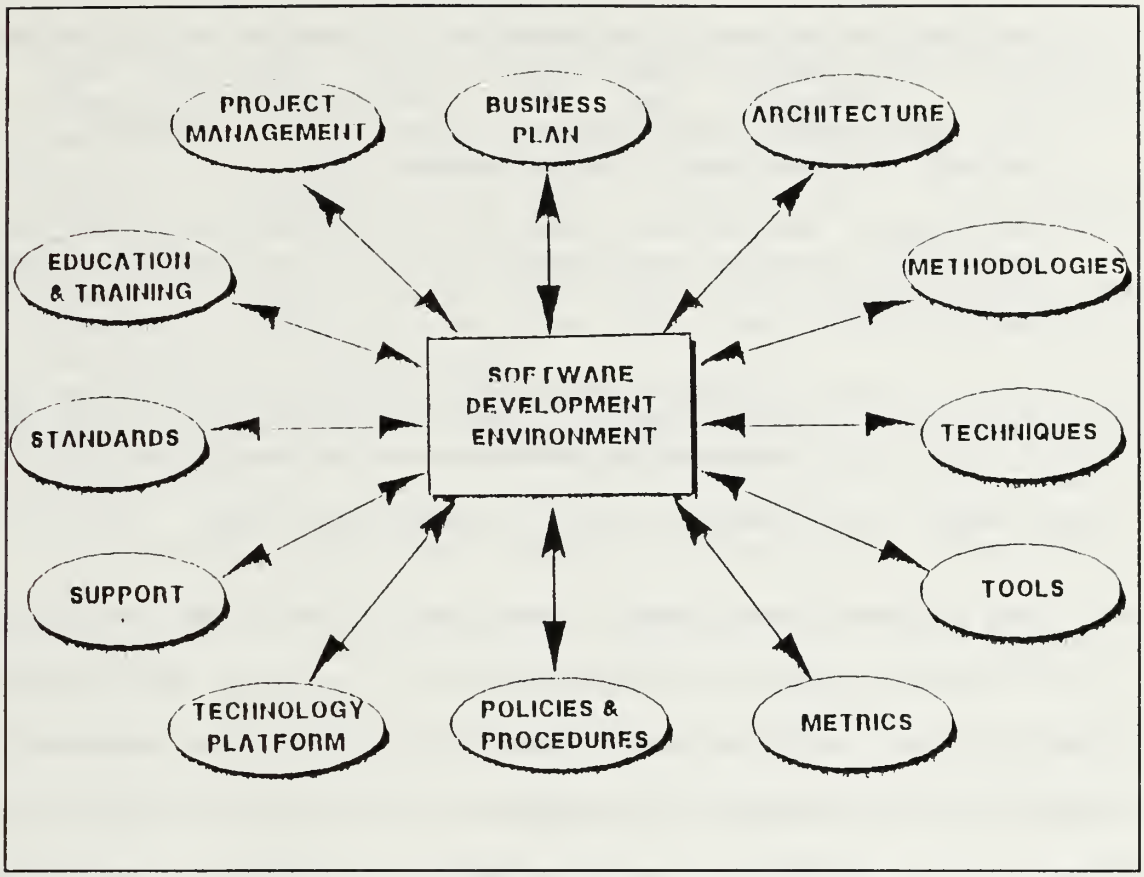


Figure 2.2 Major Elements of the Systems Development Environment (Corbin, 1991, p. 29)

The SDE is characterized by a number of factors (Jaakkola, 1991, p. 7):

- the mix of the applications portfolio including new developments, enhancements and the maintenance of existing systems
- the classes of applications such as custom development, software packages, modified software packages, etc.
- the scope of applications: corporate, departmental, work group, and end user
- the combination of old and new technologies
- the degree of end user involvement in the systems development process

- the degree to which automation is used -- the technical production environment
- the application orientation -- the type of problems to be solved and dominant problem areas
- the system development setting -- the particular mode of operation between developers and between their customers and users

To mature into a software producer, an organization requires sound business practices, an obsession with continuous process improvement, and the wise use of technology (STSC, 1992, p.1).

D. THE SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) AND ITS CRITICS

A life cycle is defined as a series of orderly, interrelated activities resulting in the successful completion, delivery, and support of an information system (CASE, 1986, p. 132). Methodologies can support one or all phases of the life cycle. Often the terms methodology and life cycle are combined especially when the methodology supports all phases: an integrated life cycle methodology.

One cycle employed in the design and development of information systems is called the Systems Development Life Cycle (SDLC). This conventional life cycle (Figure 2.3) is characterized by development with the following sequence of general activities (Agresti, 1986, p. 2):

- **Specification** -- a statement of "what" the software will do, followed by a detailed analysis of the requirements including the desired functions, performance standards, and interfacing

- **Design** -- "how" the software will meet the requirements; the structure of the software modules that perform specified functions; the data structure, software architecture, procedural detail, and interfaces
- **Code** -- implementation of the design in a programming language
- **Test** -- verification that the code executes without failure; validation so that the completed software is acceptable to the users
- **Operations and Maintenance** -- implementation and evolution of the software to meet changing needs

The common waterfall model of software development captures the major top-level phases of the software development process. Of these phases, design is emphasized. The conventional life cycle model and its variations represent a careful and systematic approach to software development by employing a series of steps in a particular order. The perceived benefit of using this structured engineering approach, including the strict controls via documentation and walkthroughs, was justified in the past for costly and complex programs.

Structured methods also appeared in the early 1970's to support the major activities of the waterfall model. These methods were a collection of procedures and concepts to increase the productivity and effectiveness of software development organizations. As such, structured techniques shifted attention from the programming phase to the front-end analysis and design phases.

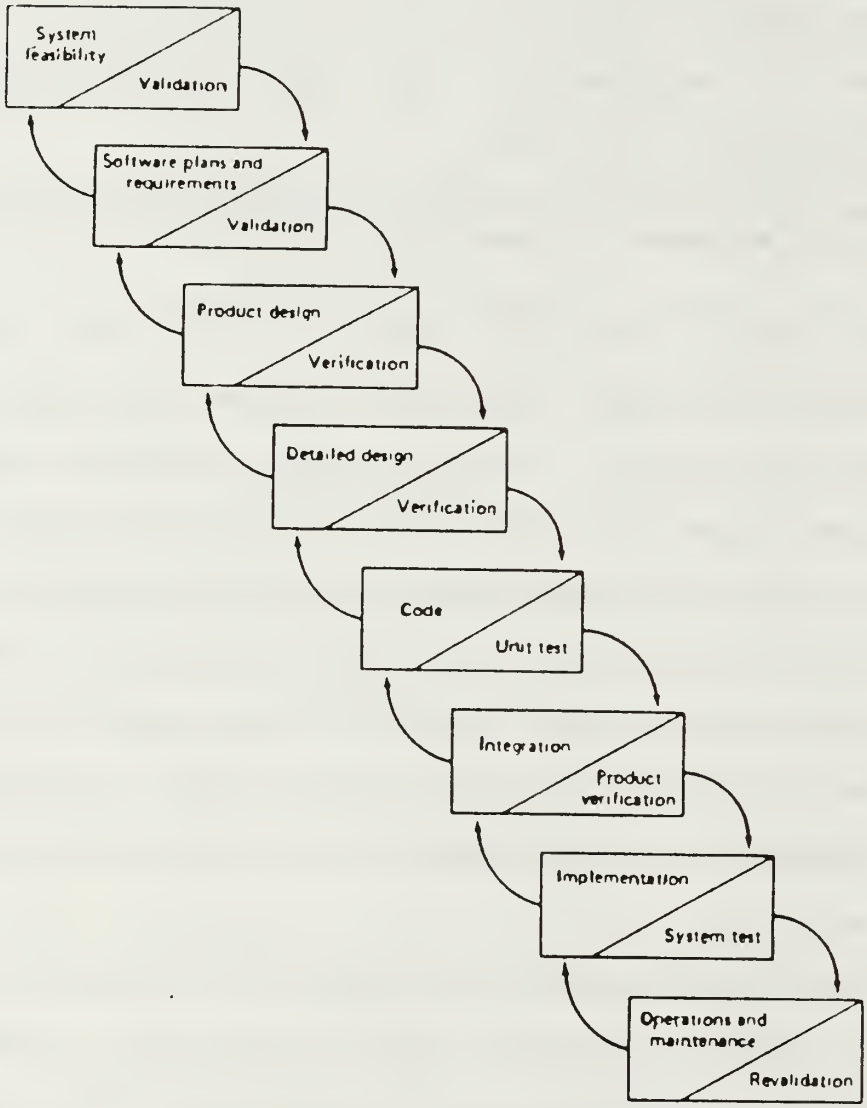


Figure 2.3 The Waterfall Model of the Software Life Cycle (Frey, 1987, p. 11)

Elements of the structured methods include (Frey, 1987, pp. 18-19):

- **structured programming** -- composing program logic from restrictive control structures: sequence, selection (if-then, CASE), and iteration (do-while); supports the construction, delivery, and maintenance phases of the life cycle
- **structured analysis** -- guidelines and graphical tools which can show the flow of data, the storage of data, and the processes that respond to the change of data; does not show control; the objective is to accurately define requirements that can be easily understood by the user; addresses the study and define requirements phases
- **structured design** -- for factoring programs into independent, highly cohesive (each module should support one and only one function) and loosely coupled modules (each module should be minimally dependent on other modules); supports the design phase and indirectly supports the construction, delivery, and maintenance phases

Structured analysis and design are companion, process-centered methodologies: analysis builds the requirements and design transforms the model into a top-down structure for programming. However, using these methods requires a tremendous amount of cross-referencing of data from one phase to the other and a lot of repetitive activity (Frey, 1987, p. 19).

It has been recognized by many that the waterfall model of the software development and its variations (not necessarily its structured methods) is dead. However, the waterfall model does provide a well known terminology base and a common

framework in which to discuss other life cycles and their associated methodologies.

One of the most common critiques of the conventional life cycle is that it represents a static, versus an evolutionary, view of the software process. For this reason, the waterfall model is primarily used when the problem and situation are well know and defined. (Frey, 1987) Unfortunately, this is rarely the case because the development process and automation can easily change the user's perception of what is possible and can often change the user's environment as well. Another criticism is that the conventional life cycle reflects the time period in which it evolved -- when software was developed by skilled professionals and computer processing time was an expensive resource (Agresti, 1986). Certainly, there were no automated support tools or techniques -- no personal computers with advanced graphics, no rapid prototyping techniques, and no local area networks, for example. Indeed, to McCracken and Jackson (1986, p. 24) "to impose this concept [the SDLC] on emerging methods in which greater responsiveness to change is possible, seems to be sadly shortsighted."

The waterfall method reflected a systematic and analytical progression of software development which deferred implementation and coding to the later phases of the life cycle. Specification was not interleaved with implementation: the "what" or the requirements of the system were separated from the "how" or the design and implementation of the system.

In other words, there is an inherent weakness in the conventional life cycle between analysis and synthesis. Physical limitations of the hardware, imperfect foresight, financial considerations, or other valid reasons can easily (and frequently do) change the behavior of a system which requires modification of the specifications. Therefore, an executable behavioral model of the system is necessary early in the life cycle to assess performance and to determine unanticipated implications and interactions of the design. Like the user that redefines the requirements when the system is demonstrated, so too does implementation change the specifications. (Swartout, 1982, pp. 26-27)

In short, active behavior promotes operational understanding. As an analogy, consider learning a new board game. It is common for the players to briefly review the rules and then start playing. The early experience of playing the game is a more effective way of learning the game than continuing to read and analyze the rules. Unlike the traditional methods that capture only the static or data structure aspects of the problem, it is also important to capture the dynamic and behavioral aspects. (Agresti, 1986, pp. 11-12)

As for top-down development, Keuffel (1991) recommends faking it! He states that logically organized diagrams are for books and for presenting your work **after** it is finished. "In the real world, information about systems is **not** often

obtained in such a prescriptive, hierarchically decomposed manner." (Keuffel, 1991, p. 39) Nevertheless, he gives five reasons why a rational design process makes sense: (1) designers need guidance when overwhelmed by a complex task (2) software development by following a process is better than proceeding adhoc (3) a standard procedure assists good design review and the transition of people from one project to another (4) having a standard process makes measuring it easier and (5) a standard process makes managerial review easier. He recommends "mining" the system using whatever procedure gets the job done and only spending enough effort as necessary and no more!

The conventional life cycle appears lacking in the following areas: it does not adequately address prototyping, end user development, uncertain and constantly shifting requirements, the interrelation of specification and design, the use of automated tool support, and the need for versatility. A new life cycle, on the other hand, would encourage a flexible development process with executable programs early in the life cycle and incorporate the use of automated tools. Or, should we re-evaluate the traditional methodologies to determine which stages can be omitted for small systems development, and determine the risks associated with such omissions? Or should management pay more attention and allocate more resources to process improvement -- to

determine how the process of software development could have been done better? (Jaakkola, 1991, p. 7)

But as Plauger (1991, p. 17) warns, "I cannot honestly report any method that will guarantee success." Most of the literature also warns that none of the new software development methodologies eliminate or replace excellent systems analysis -- a complete and accurate understanding of the problem, the requirements, and the solution. The shortcomings of the methods cannot be entirely eliminated either by automated support.

E. PROTOTYPING

Growing software demands, advances in computer hardware technology, and continuing frustrations with the time-consuming traditional life cycle process have driven software developers to pursue alternative life cycle methodologies. One of the most appropriate and practical methodologies to date is rapid prototyping. A prototype is a quick, cost-effective and controllable model that conveys the look and feel of the proposed system. Prototyping is defined as a language-independent process for building models of application systems during the software development process. Prototyping is based on the premise that users really do not know what their application should do or how it should operate: "I don't know what I want but I'll know it when I see it." (Fisher, 1987, p. 29) The practical truth is that

developers may build and test systems against specifications, but **users** accept or reject systems according to current operational realities.

Prototyping's main objective is to gain a better understanding of the users' requirements and the behavior of the system. Whereas the conventional life cycle imposed unrealistic pre-specification of the requirements, which hindered productivity, prototyping offers a more flexible, iterative approach that encourages "exercising" of the prototype, change, and experimentation. Prototyping's characteristics include that (1) it is an actual working system (2) it is comparatively inexpensive to build (less than 10% of the developmental costs) (3) it can be developed quickly and evaluated early in the life cycle (4) it can provide a physical representation of key parts of the system before implementation and (5) it usually performs only a subset of the functions of the entire system and may not have all the behavioral aspects (response time, internal control, security, etc.) of the final product. (Fisher, p. 5)

A prototype is usually built with one of two strategies, throwaway or evolutionary. In the throwaway approach, the prototype serves as the specification for its replacement. With the evolutionary approach, the initial prototype with only its essential functional requirements becomes an operational system with all the users' requirements incorporated and fully implementable.

Which strategy is chosen depends on several factors. First are the personnel resources. Second is the application and the eventual use of the prototype. Some prototypes are just explorative to determine initial requirements and functions; some are experimental to address a proposed solution before investing time, effort, and money; some are evolutionary and develop into the fully implementable system; some are used as a mock-up to determine user interfaces; and finally some are used as simulations to measure certain behavioral characteristics. The third factor is the hardware constraints -- the prototype may react differently to the load, the number of supported users, and the volume of data than the final system. Finally, the fourth factor is the availability of prototyping tools. (Fisher, 1987, p. 18-19)

The most promising candidates for prototyping are managerial systems because the business environment keeps changing and the system must react quickly to those changes. In general, though, any application with dynamic displays, user interaction, and frequent changes can use prototyping, but the decision should be weighed against the application's complexity. Prototyping according to Martin (1991, p. 109) is valuable for interactive on-line systems when the users are unsure of what they want, when the users understand the functions better than the analysts, when there is room for user creativity to improve the system, when the users do not

understand all the impacts of the new system, and when the analysts wants to elicit ideas, among other circumstances.

Prototyping usually depends on the use of automated tools such as fourth generation languages that include data dictionaries, screen formatters and painters, and report generators. These tools provide the user interface, a scheme for the organization of the data and access to the data, and the system's interface with its physical environment.

Using fourth generation languages, a developer can construct a prototype system consisting of a mixture of data entry screens, printed reports, external file routines, specialized procedures, and procedure selection menus all based on the logical database structure developed during the data modeling process. A suggested procedure for developing a prototype presented by Fisher (1987, pp. 30-31) is described:

1. Define the basic database structure derived from the logical data model. Later on, the database structure will contain test data for specific tests.
2. Define printer report formats: what data elements to print and what selection and ordering criteria.
3. Define interactive data entry screens -- the right information in the form of prompts, labels, and help messages and validate the input. Use defaults as often as possible initially.
4. Define external file routines to process data that is be to submitted in batches or created by the prototype for processing for other systems.

5. Define algorithms and procedures to be implemented by the prototype and the finished system. This may include support routines solely for the use of the prototype.

6. Define procedure selection menus. Concentrate on the functions performed as the user would perform them. This may result in combining disparate procedures into a single function executed with one command from the user.

7. Define test cases to determine if data entry validation is correct, that procedures and algorithms produce expected results, and that system execution is clearly defined throughout a complete cycle of system operations.

8. Reiterate this process by adding report and screen formatting options, corrections for errors discovered in testing, and unambiguous instructions. Suspend the process if the changes become cosmetic rather than functional.

The benefits of prototyping include (Fisher, 1987, p. 7):

1. the identification of requirements and problems early in the life cycle saving further expense and time later on; a conversational requirements tool

2. reduced development time through the reiterations of the prototype and through the knowledge gained by using it

3. the development of a system based on true versus perceived requirements

4. the ability to adapt to changes in the system's requirements

5. simplified and accelerated training by providing an operational system prior to implementation

6. a more accountable and visible system for management leading to increased communication with management

7. development effort and time is reduced by not including complete functionality

8. information requirements can be easily validated

9. the early elimination of useless functions and requirements (it is easy for users to tell you what they do **not** like)

10. a potentially increased chance of user acceptance especially if the users are actively involved in the prototyping process

There are also a number of disadvantages to prototyping (Fisher, 1987, pp. 6-8):

1. an increased tendency to skip through analysis and design which could lead to a cycle of code, implement, and repair (if there is not a specification, do not prototype); the time spent fixing the problems may exceed the time required to do detailed analysis and design

2. could lead to a design that is not flexible because it was developed too quickly

3. it is difficult to determine when to stop prototyping especially when the environment is unstable or extremely dynamic which can lead to a delayed prototype and "there is nothing worse than a rapid prototype that isn't" (Agresti, 1986, p. 6)

4. a prototype system may evolve into production before it is ready

5. end users may make unrealistic demands of the prototype and the users may eagerly adapt the prototype as the fully functional system prematurely

6. prototypes do not address the full range of operations such as security, back-up procedures, system testing, reliability and training and may not be able to handle the volume of data or perform accurate calculations

7. the actual performance and ease of maintenance of the prototype cannot be ascertained

To guard against these disadvantages, Agresti (1986, p. 6) proposes writing a prototype statement of work defining the objectives of the prototype and the range of its capabilities. Fisher (1987) proposes performing an impact analysis of any

changes to the prototype in order to justify the cost of the resources. At each iteration, he recommends determining how much functionality is present, if the design is maintainable, and whether further iterations are cost-effective.

How can prototyping be incorporated into a manageable development model or life cycle without disrupting its effectiveness or without negatively influencing managerial control of the development process? It is widely agreed that prototyping does not take the place of the entire life cycle; rather, the life cycle is supported by prototyping. Moreover, its ramifications are felt throughout the life cycle because prototyping can correctly define requirements early in the cycle which, in turn, affects design, implementation, and maintenance. It is generally agreed that prototyping should be incorporated after the analysis phase so that the problems of the information system are initially identified and potential solutions determined. One difference between the standard life cycle and the prototyping approach during requirements determination is that the computer, through the use of screens and reports, is used as the means of communication rather than paper models. A development methodology using prototyping and stressing evolutionary requirements specifications allows for the design of a flexible and usable system. Pressman's (1992) paradigm for software development begins with an abbreviated representation of the requirements; an abbreviated design specification

focusing on top-level architectural and data design rather than detailed design; the development, testing, and refinement of the prototype by the developer with the user; and finally exercising of the prototype until all requirements are formalized or until the prototype evolves into production.

F. INFORMATION ENGINEERING

Information Engineering is one of the components of James Martin and Company's enterprise engineering framework presented in Figure 2.4. The other two components are business reengineering and total quality management. The objective of enterprise engineering is to provide the tools, techniques, and task structure to implement business requirements that are resilient and responsive to continuous change and improvement. Business re-engineering focuses on the strategic vision and mission of the corporation; information engineering provides the discipline for developing integrated information systems for an organization; and total quality management focuses on the quality and standards of the deliverables for estimation and control of projects. (James Martin & Co., 1992, p. 2)

The importance of including enterprise modeling into the life cycle addresses the requirement to include the dynamic or behavioral aspects of the enterprise as well as its business rules and logic. The conventional top-down approach is based on a stable data base and a stable framework; a new

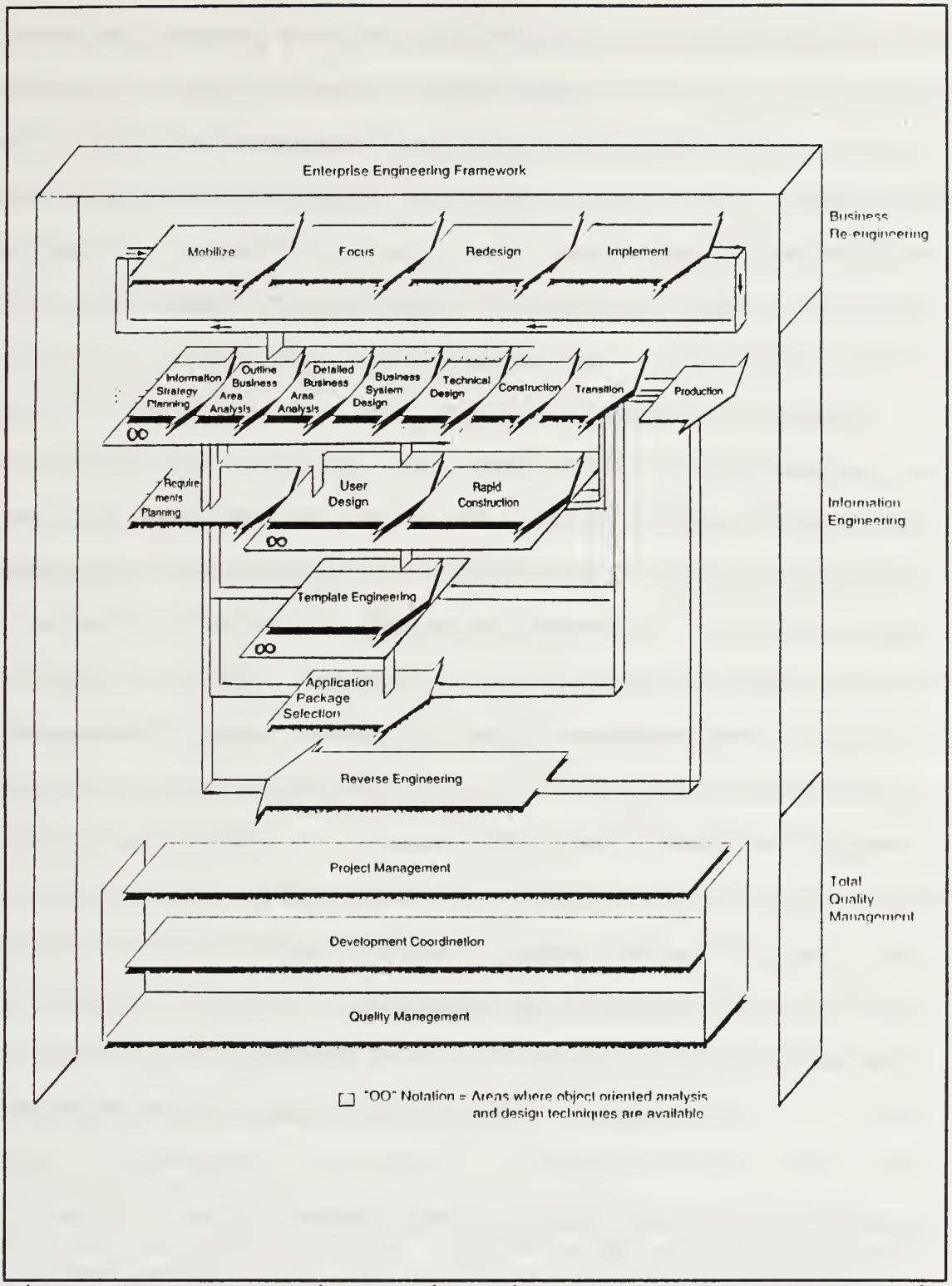


Figure 2.4 Enterprise Engineering Framework (James Martin & Co., 1992, p. 2)

methodology must model the enterprise from inherently unstable views that can only be depicted in an evolutionary and dynamic manner. Some tools fail to capture essential elements of the information system such as the rational or logic behind information flow which results in "shelfware rather than effective communications and documentation vehicles." (Due, 1991, p. 54-56)

Enterprise modeling requires that an effective methodology be supported by tools that can do more than just draw pictures. An effective enterprise modeling methodology should use one technique that consistently states the enterprise's goals, purpose, context, strategy, markets, threats and opportunities, critical success factors, controls, policies, procedures and business rules. Moreover, the technique should allow users at every level to view the organization from their perspective at any time. For example, "a financial view would allow the dynamic display of the financial implications and the consequences of changes to the views of the enterprise model by interactively modifying and executing the enterprise's financial model." (Due, 1991, p. 57) The views would be integrated by the underlying logic of the enterprise and would allow mapping of function, information, state, organization, resources, control, security, etc.[2] The rules

---

[2]One interesting approach involves developing the enterprise model as theater with acts and scenes as the functions; the actors as the subjects who manage the functions and resources; and the objects as the items being acted upon.

of the information system should be tested against the user's view of the enterprise. Other requirements for enterprise modeling include effectively recording the state and impact of the external environment, be it government regulations, the economy, or technology; being able to integrate enterprises physically distributed; and being able to incorporate technology-independent logical modeling. (Due, 1991, pp. 54-57)

Information engineering incorporates this concept of enterprise modeling which differentiates it from conventional methodologies. Information engineering is formally defined as "the application of an interlocking set of formal techniques for the planning, analysis, design, and construction of information systems on an enterprise-wide basis or across a major sector of the enterprise" or defined in terms of its primary objective, "an organization-wide set of automated principles for getting the right information to the right people at the right time." (Martin, 1989, p.1) The information engineering methodology considers information to be a strategic asset and as such should be planned, designed, coordinated and made available when needed.

Each of the actions are placed in their appropriate role and state as the "play" unfolds. As subjects and objects change, they are viewed by the audience at different roles at different times. This play-scripting or entity life history approach can provide a standard framework for capturing and displaying data, function, behavior, organization, and views of the enterprise.

Information engineering is based on an extension of structured analysis and design techniques, entity data modeling, systems integration, computer-aided software engineering (CASE), and includes other modern techniques such as rapid application development (RAD). It aims to produce working systems faster than a third generation environment and encourages strong design roles for users. Information engineering claims to be more integrated than any other system methodology by incorporating independent techniques into a cohesive concept. (Martin, 1991)

Some characteristics of information engineering (Martin 1989):

1. Information engineering applies structured techniques on an enterprise-wide basis versus a project-wide basis.
2. It progresses through a series of stages (information strategy planning, business area analysis, system design, and construction).
3. It has an evolving repository or encyclopedia of knowledge about the enterprise, its data models, process models, and system design.
4. It allows the integration of separately developed systems.
5. It is supported by the use of automated tools.
6. It encourages end user involvement.
7. It recognizes the long-term evolution of systems.
8. It incorporates the strategic goals of the enterprise.

Several of the main benefits of information engineering include (Martin, 1989):

1. the identification of strategic system opportunities which could create a competitive advantage by building supporting information systems before the competition
2. relating the data processes of the organization to its goals
3. integrating different systems--the same data is represented in different systems and the data and process models are created independent of any specific application area
4. rapid development and change of the system through automated tools
5. better control and understanding of complex systems and the interfaces between systems
6. the long-term evolution of systems
7. savings through the use of reusable design and code
8. the reduction of maintenance and backlog problems
9. a potentially highly computerized and integrated network
10. more time being spent on planning and design than on coding

How are these benefits realized by information engineering? Information engineering integrates separate data processing and decision-support systems by employing a common repository of planning information, data models, process models, and design information. It seeks to maximize the value of these systems by relating them to top-management goals and critical success factors and seeks to automate the work of building and integrating systems. It utilizes common

data entities, common rules relating to the data, and reusable design and reusable code with the encyclopedia acting as the integrator for all parts of the information engineering processes. Therefore, the objective of information engineering is to produce a set of fully structured and easily modified systems based on the common models of the enterprise and its data.

An overview of the levels or stages of information engineering and the types of diagrams used at each level are described and presented in Figure 2.5.

1. **Information Strategy Planning (ISP) --** concerned with strategic advantages, top management goals and critical success factors; a high-level overview of the enterprise, its functions, data and information needs; concerned with how technology can be used to create new opportunities or competitive advantages

2. **Business Area Analysis (BAA) --** concerned with understanding and modeling **what** processes are required to run a specific segment of the organization, a business area, and how these processes are interrelated with the data; a more restricted conceptual model

3. **Business System Design (BSD) --** concerned with **how** selected processes are implemented into procedures and how these procedures work; involves end users and automated tool support; the man-machine interface regardless of the computing platform

4. **Construction --** implementation of the procedures using code generating 4GL, end-user tools, and prototyping; a fully executable application that can be implemented in the targeted computing environment

At this point, the reader should understand the difference between functions, processes and procedures. Functions are determined during information strategic planning; represent a

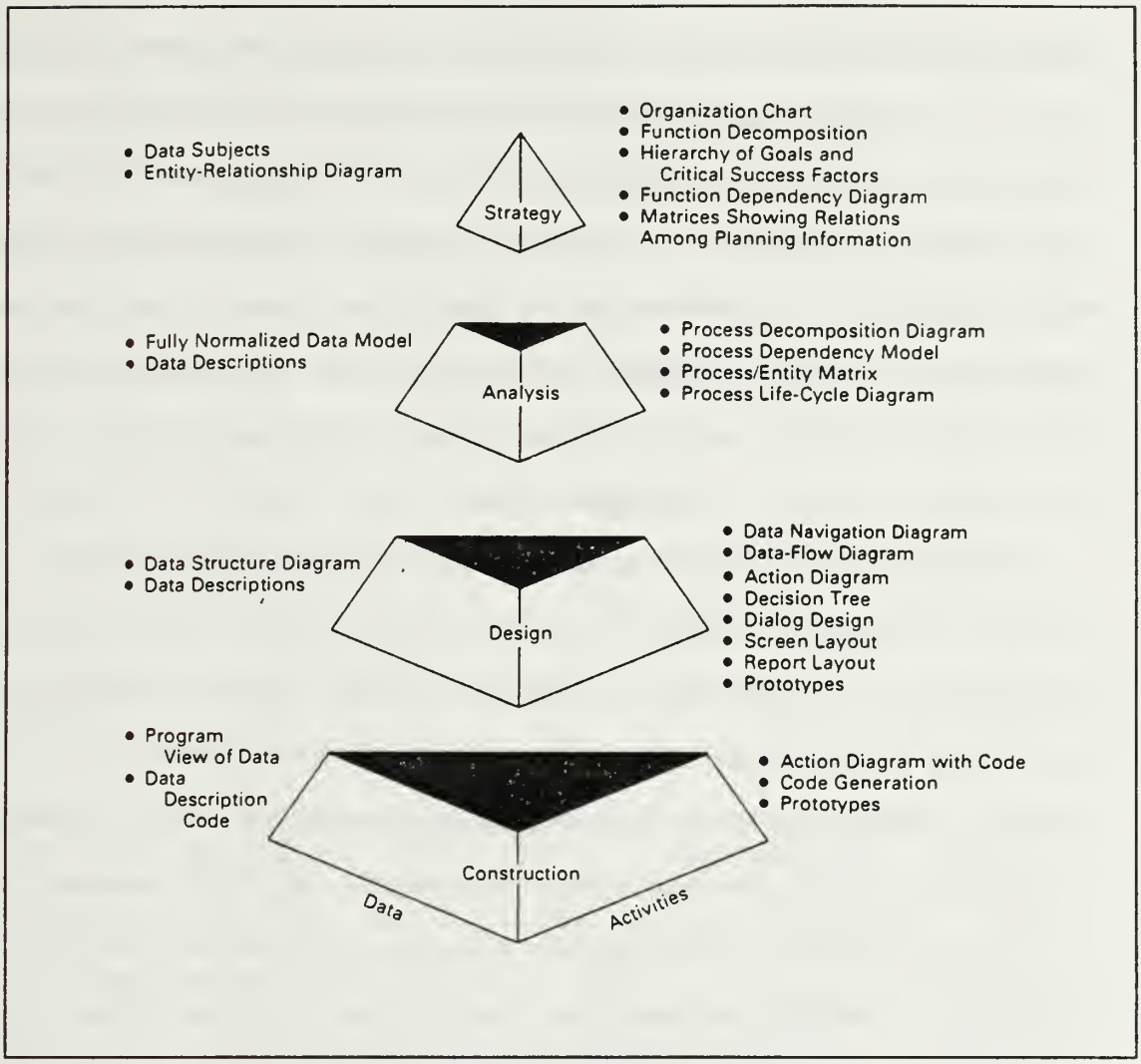


Figure 2.5 Information Engineering Stages and Tools (Martin, 1989, p. 87)

group of activities that support one aspect of the enterprise's mission; are on-going and continuous; are not based on the organizational structure; and categorizes what is done, not how. Processes are analyzed during business area analysis; are specified activities executed repeatedly in an enterprise; can be described in terms of inputs and outputs; have a definable beginning and end; are not based on the

organizational structure; and also identifies what is done, not how. Procedures, on the other hand, are analyzed during system design; relate specifically to how a process is carried out; and can change or be eliminated as technology changes. The premise of information engineering (and logical data modeling) is that whereas the procedures of an organization can change, the data, functions, and processes remain relatively stable. (Martin, 1990)

Uluakar (1991, p. 6) compares information engineering (IE) to the Yourdon Structure Methodology (YSM) which utilizes conventional structured analysis and design techniques including data flow diagrams and structure charts.

> IE and YSM life cycles are generally similar with several notable differences. IE life cycle starts with Information Strategic Planning (ISP) at the enterprise level followed by analysis of the business area of interest before focusing on a system. Business areas are defined during ISP as pieces of the enterprise which can be analyzed independent of one another. The scope of a business area should be analyzed all at once ... to avoid scope creep and future system integration problems. YSM is currently lacking a strategic planning phase. In absence of the business area concept, the YSM life cycle starts with requirements definition for a particular **system**."
>
> In addition to this difference in scope, YSM's analysis differs from IE's business area analysis in one other way. In YSM, analysis includes modelling the required processes and the flow of data in response to each event. In IE, the processes required for each event are defined during the analysis but the dynamics of the response (ie., the flow of data among the processes if more than one process is involved) is not modelled until design."

Uluaker claims that problems encountered with the structured methods are avoided in IE because data flow

diagrams are not used. For specific theoretical comparisons on each model's components, the reader is referred to his paper. It is agreed that IE and YSM are based on the same principles and that the deliverables have few differences. However, IE claims to eliminate the redundancy between data flow diagrams and the structure chart because analysis and design are integrated with IE. Changes do not have to be made to **both** the DFD and the structure chart.

Before describing the stages of information engineering it is important to discuss the so-called heart of IE, the encyclopedia. The encyclopedia is a computerized repository which includes not only the common data dictionary but also a complete coded representation of the system's plans, models, and designs. As an analogy, a file contains code for how a program will function; a repository contains information for how a **system** will function. It also provides a specific interface to control access to the objects it contains. Logical definitions of the organization are stored in a well-structured format. Usually an entity relationship data model is used for the repository information because it allows explicit definition of relationships (Bloor, 1991). The central encyclopedia also contains diagramming tools that apply rules for interlinking diagrams into a larger perspective or compound view and checks for integrity; a knowledge coordinator for checking the consistency of perspectives created by different designers; and tools for the

central analysis of the collection of information. (Martin, 1989)

This centralized planning of information is organized by subject rather than by organizational department which results in simplified data flows, more complex data structures, more consistent and accurate data, easier extraction of data when procedures change, and less maintenance work. Moreover, perspectives or views can be created by logically linking multiple screen displays and data. For example a decision tree can be connected to the structure code which can be connected to text and/or other diagrams. Rules for each diagram and for their relationships and their consistency among multiple perspectives are facilitated by the encyclopedia.

The Information Strategic Planning (ISP) stage determines how automation fits into top management's strategies and how to align system development priorities with business priorities. Note that the information **architecture** is developed independently of the current organization whereas its **implementation** reflects the organization and its concerns. The result is one of overall centralization with decentralized implementation.

The objectives of ISP are to provide top management with a view of the enterprise in terms of its goals, functions, and critical success factors, and to identify the enterprise's informational needs so that business strategy can be

translated into information strategy planning. ISP also creates an architectural framework for further analysis and design so separately developed systems will be integrated. The traditional approach of building separate systems based on each organizational unit fails to meet the integrated requirements of today's business. ISP therefore serves as the framework for implementing automated systems based on the enterprise's strategic business goals.

The top layer of ISP includes the following activities:

1. **Analysis of Goals and Problems** -- a structured representation of the goals and problems of an enterprise with their associated organizational units, information needs, and systems.

2. **Critical Success Factor Analysis** -- identifies those areas that are critical to the success of the organization; identifies critical assumptions that require monitoring, critical information needs, and critical decisions.

3. **Technology Impact Analysis** -- examines the business opportunities and threats caused by advanced technology and its potential impact on services, changes in corporate structure, new products, etc.

4. **Strategic Systems Vision** -- strategic opportunities for creating new systems in order to be more competitive; may require restructuring rather than automation

The second layer is concerned with modeling the enterprise and includes:

1. **An Overview Model of the Functions of the Enterprise** -- maps the business functions hierarchically; associates the business functions with the organizational units, locations, and entities through computerized matrices

2. **Entity-Relationship Modeling** -- creates a chart of the entities and their relationships; an overview of the data stored in the enterprise databases. Entities are associated with business functions in a matrix and the matrix is "clustered" to determine business areas.

In review, the enterprise model creates an overall framework for future detailed analysis. It consists of an overview of the entities in the enterprise, a decomposition of the business functions, and a matrix mapping entities against business functions in order to proceed to business area analysis. (Martin, Book II, 1990)

According to James Martin (1990, p. 184) a business area is clear-cut with definable boundaries; is small enough to allow business area analysis, but large enough to take advantage of a shared database in a naturally coherent way; has no overlap of function with other business areas; and is generally **not** updated by other business areas, although data can pass between business areas.

A business area is defined as sufficiently bounded and constrained when (1) the accessed data (2) the processes including their timing and coordination (3) the business relationships with all their intricacies and (4) the business rules and policies affected by the processes and flows are all well known and clearly defined (Haas, 1991). Texas Instruments claims that business area analysis can proceed without an ISP although Haas (1991) recommends **not** to bypass the ISP for **unbounded** systems.

To determine which business area to develop first, the following factors can be used to rank the projects (Martin, 1990).

- **Potential Benefit** -- return on investment including tangibles and intangibles; achievement of critical success factors; achievement of goals; solution to serious problems; the competitive impact
- **Demand** -- business urgency; pressure from senior management; assessed need; political overtones; current management priorities
- **Organizational impact** -- number of organizations and people affected; whether organizations are geographically dispersed; qualitative effect
- **Existing systems** -- adequacy or value of existing systems; relationship with existing systems; estimated future costs of maintenance; operational costs; automation potential
- **Likely success** -- complexity; degree of business acceptance; length of the project; speed of implementation; prerequisites; risks; project staff availability and expertise

The objectives of business area analysis (BAA) are to provide a more precise and clear understanding of a business' data and activities (functions, processes, and procedures) and their interrelationships. BAA refines the information architecture model defined during ISP. Specifically BAA (TI, Guide to the IEF, 1988, p. 113):

1. Identifies and defines the type of data required.
2. Identifies and defines the business activities of each business function.
3. Defines the data required for each business activity.
4. Identifies the necessary sequence of business activities.

5. Defines how business activities affect the data.

6. Produces a plan for business system design. Normally several business systems will support a single business area.

BAA creates a fully normalized data model for application design and construction; a model of the business activities and their interdependencies; and a link between the data and the processes by identifying the data used by certain processes. Figure 2.6 presents the classical data and process modeling techniques used. These tools and techniques are explained in detail in Chapter V for the IEF integrated CASE toolset.

1. **Data Model Diagram** -- a fully normalized data model is built for the business area; an extension of the entity-relationship model created during information strategic planning

2. **Process Decomposition Diagram** -- the business functions are decomposed into lower level processes and a tree-structured decomposition is produced

3. **Process Dependency Diagram** -- also referred to as a process flow diagram; maps the dependencies of processes: process can only be executed after another is created; shows the data flows from one process to another but does not show the contents of the data

4. **Process/Data Matrix** -- maps the processes against normalized data, showing which processes create, read, update, or delete the data; ensures data and processes have all been determined and the process dependencies have been assessed correctly.

By defining common processes, non-redundant data modeling can be achieved. Unlike many older, independent systems, which usually had the same data defined differently in different

places often with different names, information engineering instead represents all data in one encyclopedia and creates different user views.

During process modeling, entities, processes or procedures may apply beyond the business area or span many business areas. Therefore, it is important to enforce the integrity between business areas. Such analysis checks include data flow connectivity (all flows are continuous and connected to valid sources); data flow course analysis (determines the path of data flow regardless of how many levels the data traverses); data conservation (input must equal output); data model completeness (all processes must be represented); and process model completeness (a process must create or terminate at an entity and all entities must be read or updated). It is also necessary to identify key decision-making processes. What decisions should be made, where should they be made, who should make them, who depends on the decisions, what information is required for the decisions, and when or how often should the decisions be made?

The final step in business area analysis is to prioritize the projects identified during business area analysis and determine where system development effort should be utilized. The factors previously presented for prioritizing business areas can be used. As a result of this analysis, current procedures may be eliminated or changed and high priority requirements identified.

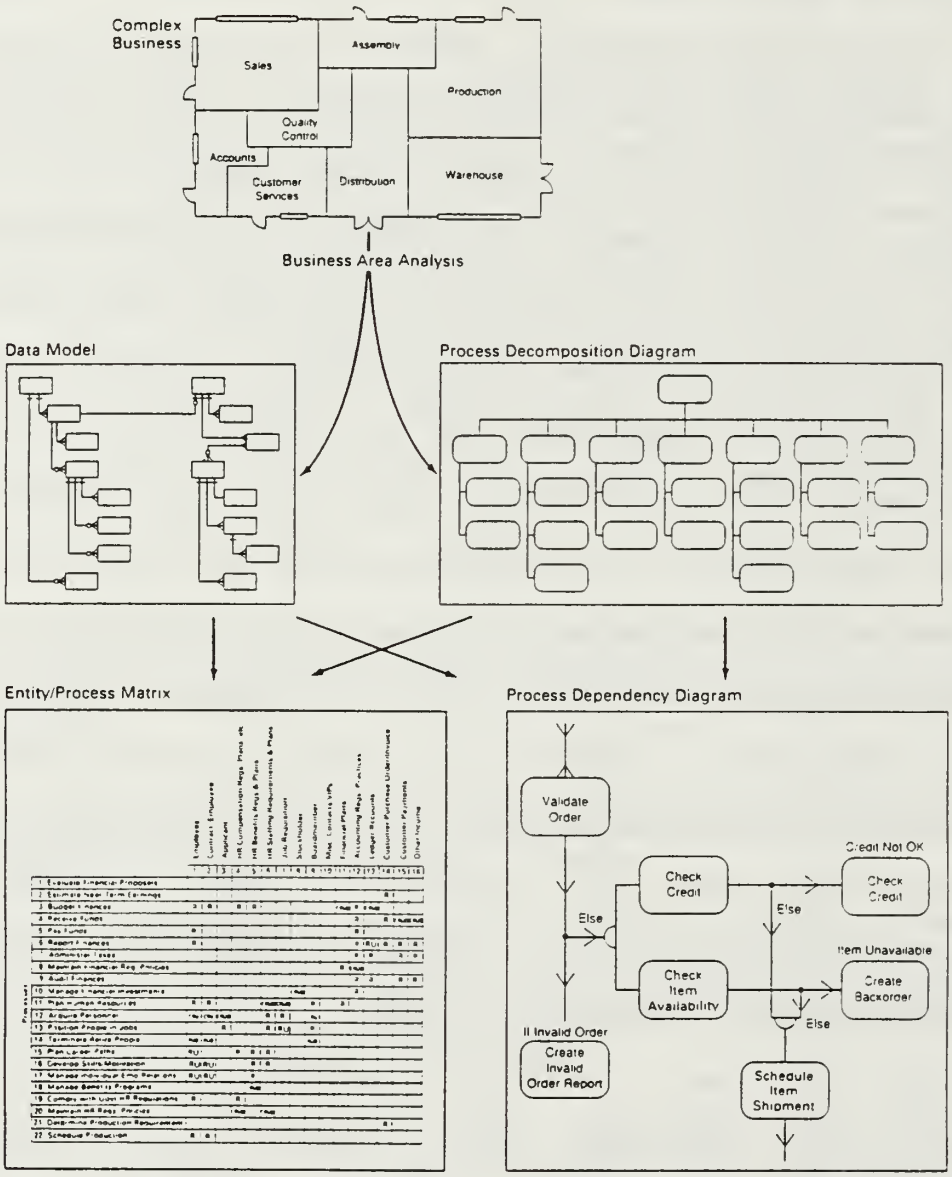


Figure 2.6 Business Area Analysis Diagrams (Martin, 1990, p.200)

As a result of business area analysis, the developer usually knows which processes to implement and in which sequence.[3] This information is extracted from the data and process models into a design workbench which has tools to facilitate prototyping and rapid application development. The design workbench then drives a code generator. The end products of the code generator are database code, test data, job control code, and documentation. Thus, the purpose of design and construction is to accurately translate a customer's requirements into a design in sufficient detail that it can facilitate the generation of code.[4] The term integrated CASE (I-CASE) is used when the planning and modeling tools are integrated with the design tools and the code generator, all using the same encyclopedia. (Martin, 1990).

In the past, the tools for design were usually based on text specifications which took time to develop; were difficult to visualize; were prone to errors, omissions, and ambiguities; could not be checked by a computer; and could not be used as input to a code generator. Note that these tools do not have to be linked to information engineering. Some of

---

[3] Design does not necessarily have to wait until BAA is completed -- it can be retrofitted if necessary.

[4] Reverse engineering proceeds somewhat in the opposite direction: from unstructured to structured code (restructuring)and from code to redesign (reverse engineering). Other functions can be added to produce new code for the improved system. (Martin, 1989)

the tools are illustrated in Figure 2.7. (Martin, Book III, 1990)

1. **Decomposition diagrammer** -- provides a high-level overview statement about a design to be successively decomposed into finer detail
2. **Action Diagrammer** -- facilitates the building of structure procedures and structured code
3. **Data flow diagrammer** -- shows the flow of data among modules of procedures or programs
4. **Data model diagrammer** -- although used in BAA, this tool allows portions of the overall data model to be extracted for use in the design stage
5. **Data structure diagrammer** -- allows appropriate parts of the data model to be represented as structures used by a particular database management system
6. **Screen painter** -- allows quick screens design for computer-user dialog
7. **Dialog generator** -- links the screens for the user interface
8. **Report generator** -- allows the structure and layout of a report can be created quickly along with calculations of derived fields
9. **Database code generator** -- generated database code directly from the data structure diagram
10. **Code generator** -- creates executable code from the highest level specifications possible
11. **Test data generator** -- creates testing aids to generate test data and facilitates a sequence of testing steps

These tools are not the answer to good design because a designer can still produce incompatible, fragmented, and poorly designed programs, although probably faster. The generated code is only as good as the data obtained from prior

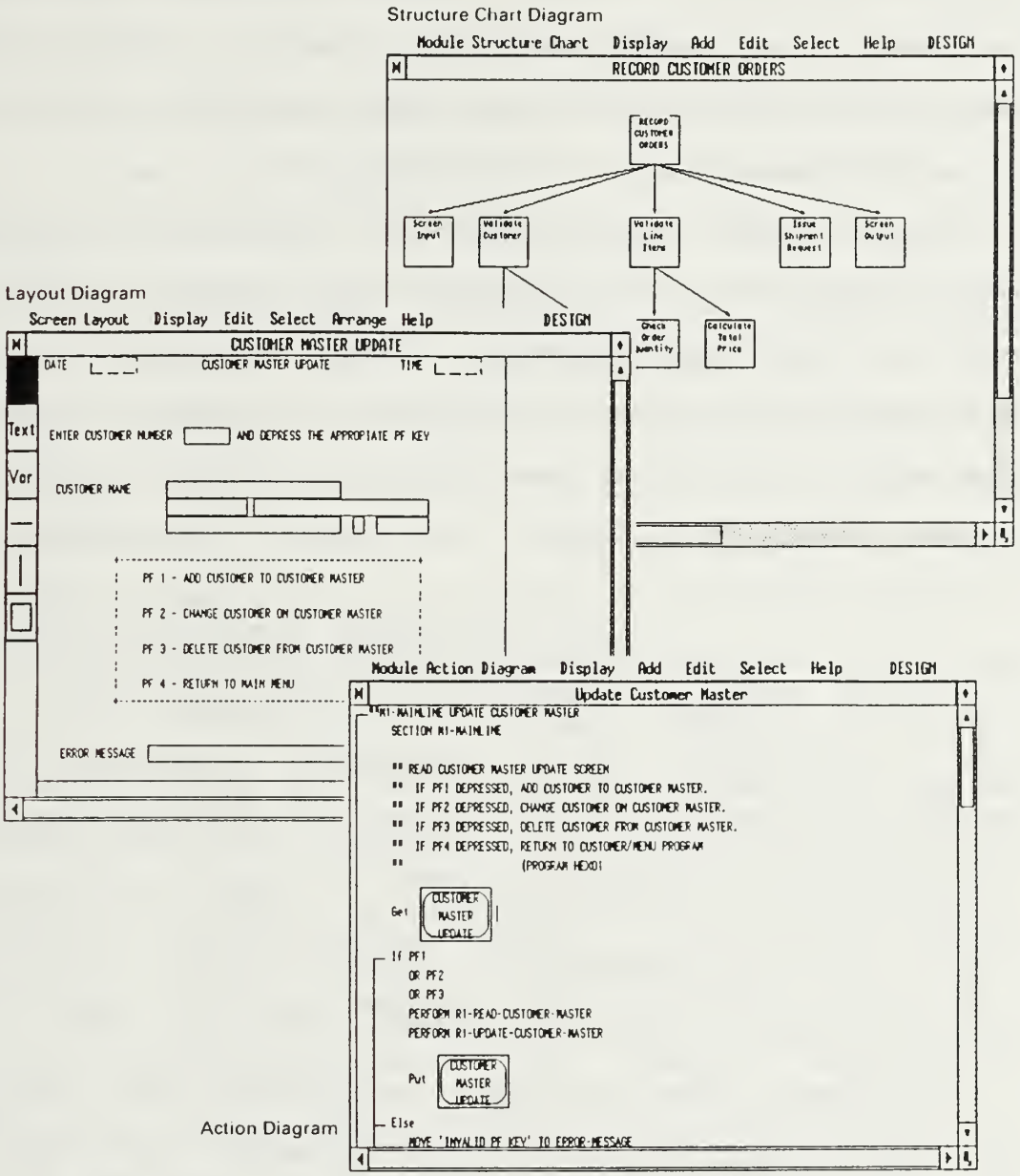


Figure 2.7 Some Diagrams used During Business System Design (Martin, 1989, p. 115)

information engineering techniques and the resulting system is only as good as its design. Sound information engineering techniques for design must be applied. This includes developing standards, using reusable components, and creating an architecture that is flexible to change.

During design and construction, the designer determines, among other things, what procedures are required to implement certain elementary processes, whether multiple operations should be combined into one procedure or combined with other procedures, and how the user interface (screens, reports, the layout) should be designed. For example, should the design be highly structured to guide the user or dynamic so the user can direct the system based on shifting priorities? Should there be two levels -- one for the frequent user who may want streamlined commands and minimal text and one for the beginner who wants extensive help and descriptive text? What commands, function keys, and display properties (prompts, reverse video, etc.) should be standardized in order to create a consistent user interface? (Texas Instruments, Book III, 1990)

To answer these questions, the designer must understand the data involved, the activities performed, the interaction between the data and the activities, as well as the underlying information architecture and last but not least, the user's environment. Design in the information engineering methodology emphasizes end user participation in the design

process itself through workshops such as joint application design and prototype reviews.

Maintenance can be performed, not by changing the code, but by changing the design followed by regeneration of the code. It may also be possible to optimize performance by modifying the design. The final design step is perform a technical analysis to determine the implications of implementing procedures on certain equipment or with a certain language. The last two stages, transition and production, involve the installation of the new system into its production environment.

G. RAPID APPLICATION DEVELOPMENT (RAD)

Step-by-step software development is being replaced by rapid application development (RAD) techniques, a more intuitive approach that involves constant interchange between developers and users at every stage of the development and focuses on producing systems quickly. The users and analyst define the screens and reports, the data design, the flow of control, and the program's logic. RAD is employed for projects with requirements that are difficult to specify in advance and that do not use complicated algorithms. RAD boasts that it can increase development speed and quality at a lower cost. Just as information engineering offers long term benefits for the future, RAD offers fast development for today.

It builds on the information engineering methodology and techniques. A comparison of the IE methodology and the alternative RAD path is presented in Figure 2.8. RAD, defined in the IE context, consists of two IE phases: (1) System Planning and Design and (2) Construction and Cutover. Information Engineering, in its entirety, is therefore condensed into three phases: Information Strategy Planning, Business Area Analysis, and RAD (Figure 2.9).

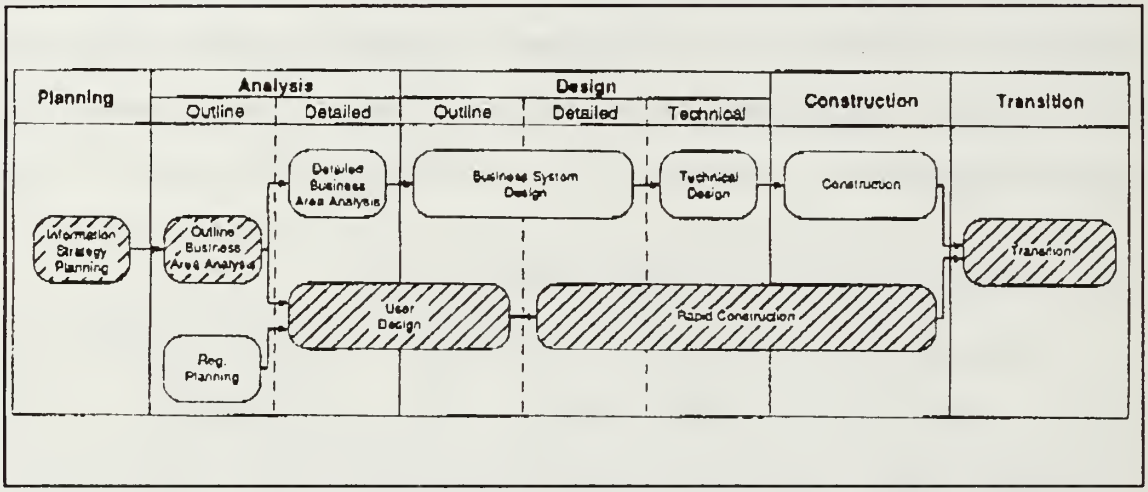


Figure 2.8 Traditional IE versus the Alternative RAD Path (James Martin Associates, 1992)

RAD usually involves a small team of information experts, a rapid prototyping capability, automated tool support such as CASE, reusable code, incremental development, integrated joint application development (JAD) and a rigid time line. Incremental development is defined as dividing the project into small and manageable pieces so that each can be analyzed, developed, and delivered in a short time, usually a few months. "RAD exploits the 80-20 rule; 80% of the value of an

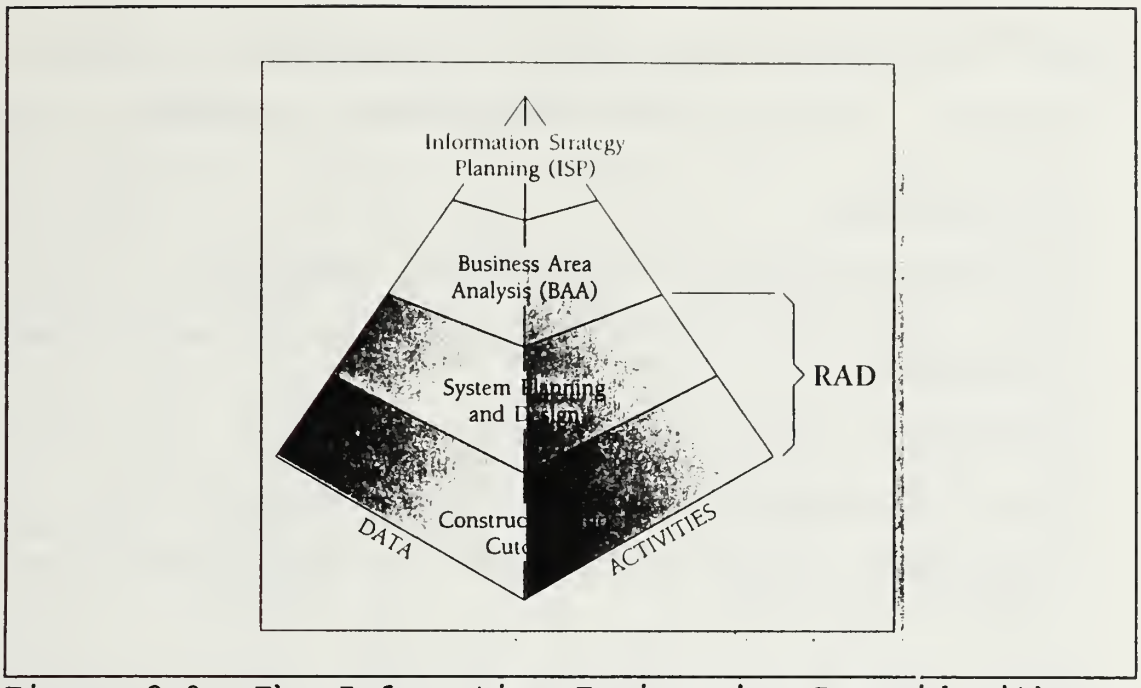


Figure 2.9 The Information Engineering Pyramid with RAD (Martin, 1991, p. 351)

application can be achieved with 20% of the application. RAD identifies and delivers the essential 20%." (Merlyn, 1992, p.9) Developers and users benefit from the initial system: the users' experiences with the initial system assist the developer for the next phase. As a result, the final application is closer to the user's solution, not the programmer's solution.

RAD as formally specified by James Martin (1991) is a developmental life cycle used to develop systems faster (months versus years) and of higher quality at a lower cost with fewer people. To Martin, the four essential elements for

rapid development are tools, a methodology,[5] people, and management. RAD in a modern environment integrates:

- prototyping;
- graphical computer aided modeling and design;
- a repository of design information and reusable components;
- automation for enforcing design integrity;
- an integrated code generator and testing tools;
- thorough end-user interaction with developers aided by tools

I-CASE tools go hand in hand with RAD because they offer computerized precision, detail, integrity, and fast development through technical design and code generation.

Martin (1991) stresses, however, that there is no compromise between quality (defined as meeting the users' requirements as effectively as possible upon implementation) and speed of development. Burden (1991) warns that introducing "rapid" to application design may mean hurried and he needs proof that RAD serves to increase quality. Does rapid really mean developing systems better so that it takes less time? RAD should also fit into a planned infrastructure so that systems can be integrated, taking advantage of shared data and reusable designs. Isolated RAD should be avoided.

---

[5]As for the methodology, James Martin and Co. offer RAD Expert, a computerized hyperdocument methodology which states what is needed at each task, how to succeed at each task, and even what to avoid so things don't go wrong.

None of the techniques of RAD replaces good and complete analysis efforts. Vaughan warns of "RAD Trap":

> RAD presents a potentially massive trap for the unwary, and thus the proclivity to create a portfolio of applications that were never conceived to work together except in a highly complicated, evolutionary fashion--without the benefit of an overall architecture. In the last two decades of application development, the pendulum has swung several times from increasing to decreasing rigor and formality, and from strategic to tactical approaches. The return to well-defined development processes, with emphasis on early life-cycle phases and the use of rigorous graphics techniques, makes RAD possible. Enterprise modeling...helps define RAD projects and ensure a shared information resource environment. RAD is not an alternative to CASE disciplines---it builds on them. Those who see RAD as "seat-of-the-pants" development have missed its most critical aspects and will find themselves creating bad applications rapidly (or creating good applications without infrastructure, leading to bad systems and high maintenance overhead). RAD only makes sense after CASE methods, JAD techniques and disciplined software processes have been established. (Vaughan, 1992, p.9)

RAD according to James Martin (1991) is divided into four phases: requirements planning, user design, construction, and cutover or transition. Requirements planning determines the functions of the system and the business objectives to be solved. A joint requirements planning (JRP) workshop can be designed so that all users jointly establish the requirements and detailed functions for the system. I-CASE tools and prototyping are commonly used with the computerized repository serving as the input. The second phase, user design, determines the nontechnical design of the system: the data and process models, screen and report designs, detailed designs and rough prototypes, again using the existing repository as

input. Usually two joint application design (JAD) sessions are conducted: the first for the initial design and the second for prototype review. Inconsistencies, incomplete data, or ambiguity can be detected instantly with CASE tools. An example of a typical analysis performed during a JAD workshop includes:

- determining what the steps are in the procedure
- building an initial flow diagram showing the steps
- examining each procedure step in more detail
- for each procedure, create a partial prototype
- address unresolved issues

At the end of the second session, the construction team becomes involved to solidify the prototype designs, to divide it into subsystems if necessary, and to test the procedures. Its physical design and configuration compatibility with existing hardware are also taken into consideration. Outputs of the construction phase include coded database descriptions, executable and optimized program code, and technical documentation. The last stage is to cut over to the new system which may necessitate additional training, organizational changes, and parallel operations of the manual system.

Martin (1991) also advocates the use of SWAT (Skilled with Advanced Tools) teams that join the RAD project at the first JAD workshop. These small yet productive teams take the

output from the JAD workshop and complete production usually within a three month time period. Some favorable factors that have led to SWAT team success include:

- a small, high quality, and highly motivated team
- a contractual wall around the project
- an enthusiastic user department able to respond to questions fairly quickly
- excellent database administration support

Unfavorable factors included a lack of continuity in user involvement from BAA to system design and a BAA model that did not capture all the business logic. (Martin, 1991)
Note that the use of CASE was assumed but was not listed as one of the most important factors except in the context of having a complete requirements analysis.

A variant of RAD referred to as timebox methodologies also warrants discussion. Like RAD, a core system is built quickly with refinements added successively; but with the timebox methodology, a working system must be delivered at an immovable deadline. This methodology is justified because it is better to have a limited system functioning in a short time than to have to wait for a comprehensive system later on. Such limited functionality must not sacrifice quality -- the system must be built to be changed and enhanced quickly. It is not surprising then that timebox methodologies employ evolutionary prototyping and code generation.

H. A TAILORED AND UNIVERSAL METHODOLOGY

As can be ascertained from the preceding discussions of the various methodologies, each has its own conceptual development framework and philosophy. Each promises the same benefits of general applicability and overall usefulness. In search of an megamethod, McCracken and Jackson (1981, p. 23) states "to contend that any life cycle method, even with variation can be applied to all systems development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous." [6]

Agresti (1986) does present a framework for a flexible development process. Its elements are activities such as interviewing users or prototyping a system; intermediate products such as display menus; control points such as a demonstration of a system's initial capabilities; and baselines such as a set of products representing a version of the system. The manager defines these elements for each project which can vary from project to project and also defines what is meant as progress for specific control. The choice of elements is influenced by certain process drivers such as the experience of the developer with the application

---

[6]This search for an umbrella methodology has already been undertaken by the European Commission (EC) in an effort to not only address differences in the techniques, definition and natural languages across Europe but also to serve as a tool to help the Commission to judge software development proposals. (Johnston, 1991)

area and the software product; the availability and effectiveness of software support; the interfaces involved; the extent and level of end user involvement; the degree of requirements understanding by the user and the analyst; and the operational characteristics of the software. Such an effort understandably involves more managerial involvement but also allows flexibility.

A new acronym has entered the literature to describe a global methodology that can handle all types of system development situations: ASDM or the Advanced Systems Development Methodology. Universal is defined as "covering the entire scope of a system's development process with a management perspective; providing detailed and comprehensive development methods; and being applicable to all systems development situations." (Jaakkola, 1991, p. 8)

A discussion of ASDM's requirements follows. The ASDM must be flexible and contain a defined set of methods based on all three approaches to software development: traditionally structured, automated, and prototyping approaches. The proven foundations of systems analysis and design should be retained and supplemented by prototyping. Such iterative development can be made possible by using automated tools that facilitate the cyclic refinement of systems requirements and the regeneration of working systems. The project management component must include not only the business needs of the system but also the central factors that relate to the success

of the project: end user satisfaction, senior management participation and commitment, quality control, and risk management. Standards must enhance the quality of the work, not restrict it. Finally the methodology should aim to be self-documenting and evolutionary.

Jaakkola (1991) also suggests a tailoring method to streamline the ASDM such that it reflects the characteristics of the system development situation and facilitates the managerial control and execution of the project. It is also important for the organization to fully identify the range of its development activities. Certain projects may require a combination of the step-by-step, automated, and iterative approaches. For example, a data driven approach works well when the organization supports the concept of a corporate data model and has grouped business functions; otherwise, a process-driven approach especially when information requirements are not clearly defined may be necessary. The result is a project specific methodology with its own set of development techniques, supportive automated tools, and standards and documentation established not externally, but by the project team itself (Jaakkola, 1991).[7]

---

[7]Foresight, A CASE resident systems development methodology claims to allow the user to customize the methodology, or even overlay the current systems methodology on top of Foresight. It also allows the user to create a standard process against which to measure and manage projects.

III. SOFTWARE DEVELOPMENT TOOLS

A. RELATIONSHIP OF A TOOL TO ITS METHOD

It is important to distinguish between a tool or technique used by a methodology and the methodology itself. Which comes first? In theory, many claim that an assessment of the methodology should occur separately from and prior to an assessment of the tools that support the methodology.

In practical terms, though, an automated tool may provide an enabling technology to successfully employ the methodology, and as a result, may reduce development time, improve communication, and improve a project's cohesion, maintainability, and supportability. CASE tools may also enforce a standardized development practice, enable reverse engineering, and provide more consistency between specification, design, and code. The justification for CASE was built upon the need for standardized and integrated software development methods. Often, the methodology is packaged with the tool (although some tools do allow customization of the methodology) with the methodology providing the infrastructure for controlling CASE techniques. CASE (1986) states that it is probably easier to adapt the methodology to the tool because the methodology or life cycle is more flexible than the tool.

Notwithstanding, one should avoid selecting a methodology simply based on the tools that support the methodology; otherwise, an organization may be stuck with a product that does not meet the organization's goals or systems' requirements. Therefore, it is recommended that an organization should first focus on the methodology, then examine the tools, and select both together. (Teledyne Brown Engineering, 1988) In an April 1991 survey, 44% of 143 respondents chose the methodology before the tools, 38% chose the tools first, and 14% chose both together. (Sullivan-Trainor, 1991)

B. FOURTH GENERATION LANGUAGES

Fourth generation languages (4GL's) and techniques represent a class of programming support tools. They allow the programmer to represent structures at a high level of abstraction -- at a level close to natural language -- by eliminating algorithmic detail and machine instruction sets. By using fourth generation languages, the programmer can concentrate on the business functions of the application rather than on the intricacies of coding. Complex functions can be executed with few commands. As opposed to fifth generation languages, 4GL's usually do not contain artificial intelligence components such as inference engines and expert and knowledge-based systems.

Fourth generation languages are also referred to as nonprocedural languages because they specify **what** is being accomplished without describing why. For example, the nonprocedural statement "list by customer average (invoice total)" does not include instructions on how to sort the list, compute the average, or determine how the page should be formatted. (Martin, 1985)

Fourth generation languages were created to alleviate the problems associated with third generation languages (COBOL, PL/1, ADA, etc.). They were designed to speed the application development process, by avoiding alien syntax and mnemonics; they were designed to make applications quick and easy to change thereby reducing maintenance costs; they were designed to minimize debugging; they were designed to generate error-free code from high-level expressions based on the requirements; and they were designed to make languages user-friendly so that end users could do their own programming.

Fourth generation languages generally have the following properties although they vary greatly in their power and capabilities (Martin, 1985):

1. They are user-friendly.
2. A nonprofessional programmer can obtain results with limited training.
3. They usually employ a database management system.
4. Nonprocedural code is used wherever possible.
5. Default assumptions are made wherever possible.

6. They are designed for on-line operation.
7. They enforce structured code.
8. They are easier to maintain and easier to understand than third generation languages.
9. They are designed for easy debugging.
10. They are able to produce and modify prototypes quickly
11. They usually have syntax-directed editors to edit the input before the commands are processed.
12. They have a smaller set of commands as compared to third generation languages.

Fourth generation languages can vary from being merely query languages, report generators or graphics generators; others can create complete and complex applications. It is not surprising, then, that the major components of 4GL's for routine applications include administrative functions for cataloging procedures; a data specification facility to design files or employ data previously defined; a report generator and/or screen painter; a dialogue facility for user-computer interaction; a rule specification capability to define conditions or decisions; and an overall procedural facility to specify the structure of the program (loops, conditions, nested routines, etc.). Figure 3.1 presents the components of an ideal 4GL. Fourth generation languages also require an infrastructure to carry out these functions such as multi-user access, security and recovery features, among others. (Martin, 1985)

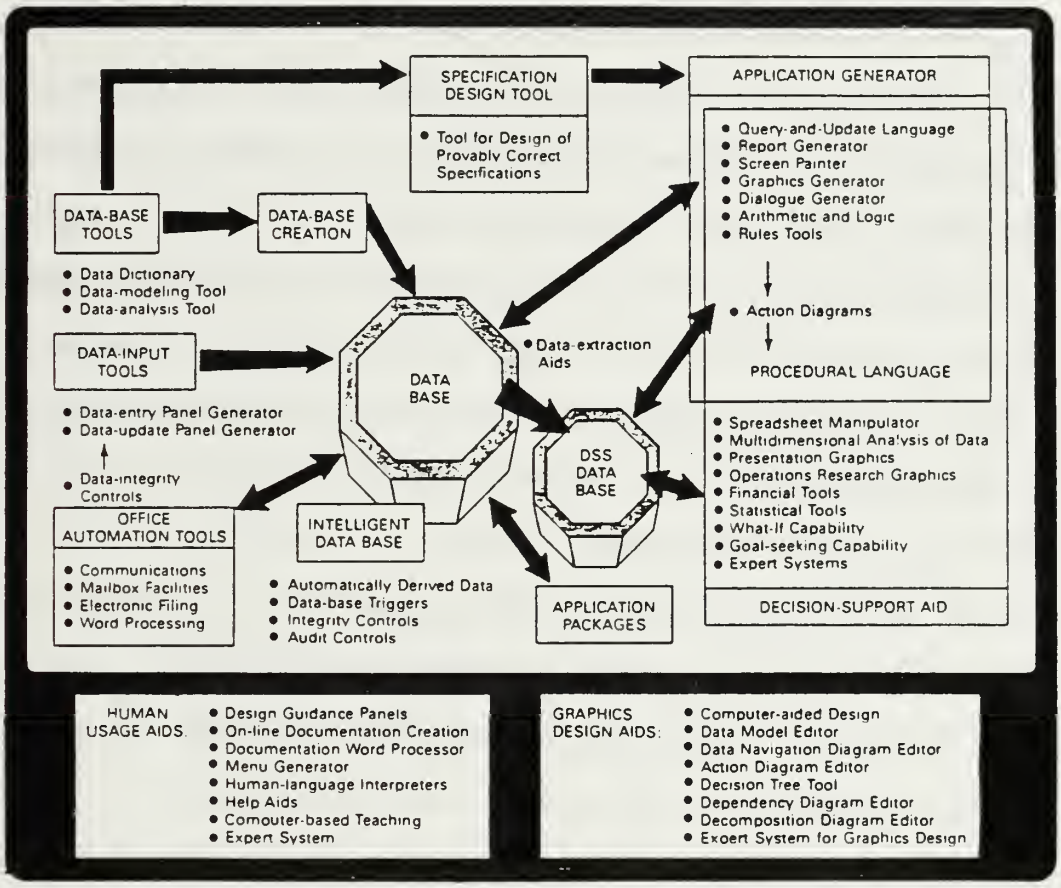


Figure 3.1 An Ideal Fourth Generation Language Facility (Martin, 1985, p. 369)

Many 4GL's are dependent on their data dictionary or encyclopedia. Their command set may be domain-specific -- they are designed for only a specific class or range of applications. Unlike third generation languages, 4GL's cannot

be applied equally to all software applications: one selects the language to fit the application.

Pressman (1992) states that 4GL's are limited to business information systems applications, specifically to information analysis and reporting that is keyed to large databases. The time required to produce software for small and intermediate applications may be reduced, but for large software development efforts the time and effort in analysis, design, and testing obscures any savings through 4GL's. Opponents to 4GL's claim that the code produced by 4GL's is inefficient and that the maintenance of large systems developed by 4GL's is open to question. (Pressman, 1992)

Some reasons why application programming is **not** done with 4GL's include: high machine resource requirements, database incompatibility, previous costly investments in non-4GL installed systems, and limited functionality. Fourth generation language programs are usually limited to low-volume inquiry and update systems.

Fourth generation languages do have distinct advantages; they offer speed, flexibility, and ease of use. Fewer programming instructions need to be written; programs can be created, modified and enhanced faster; end-user training is reduced; the complexity of developing on-line and database inquiry programs is less; and the level of programming expertise is less than other languages. In short, 4GL's reduce programming time. (CASE, 1986)

Prototyping is often interchanged with software development using a 4GL. CASE (1986) states that 4GL's, in and of themselves, do not improve program design; however, prototyping with a 4GL can **support** analysis, design and programming but again they are not substitutes. These tools need to be invoked in the context of a software development methodology to validate design concepts: a functional prototype is of no value if there is no data integrity or integration.

A 4GL methodology for software development is shown in Figure 3.2. It represents interactive application development through prototyping with a 4GL. It consists of a requirements gathering phase, a design strategy phase for larger projects, and then an iterative prototyping phase. The programmer must also perform thorough testing, document the application, and

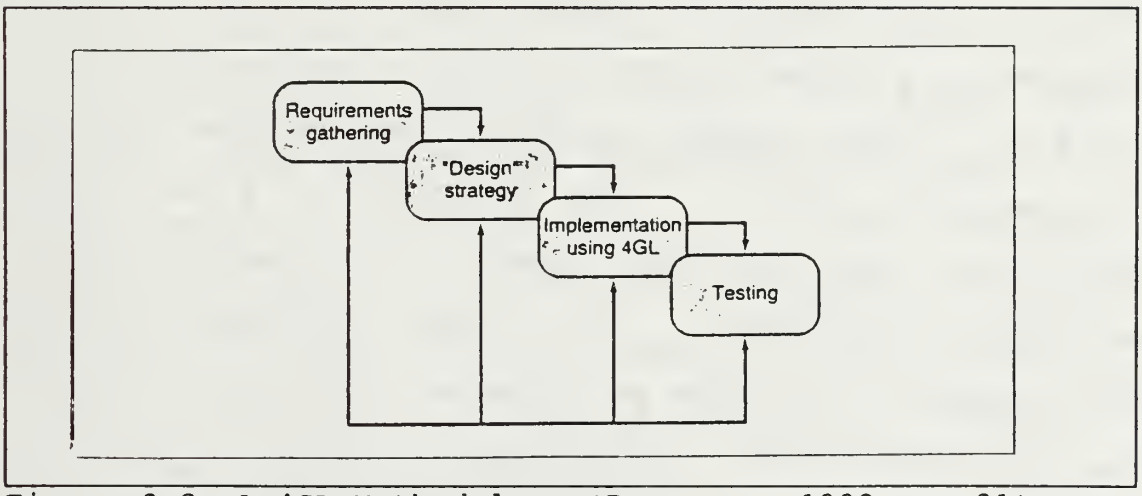


Figure 3.2 A 4GL Methodology (Pressman, 1992, p. 31)

plan for transition to the system for successful implementation. (Pressman, 1992)

C. FOCUS

FOCUS, from Information Builders Inc. (IBI), is a versatile fourth generation language introduced in 1975 for the IBM mainframe. Since then, FOCUS now runs on MS-DOS, OS/2, and major LAN's as well as DEC, VAX, and UNIX platforms, among others. FOCUS offers a non-procedural alternative to traditional development methodologies. It is installed in over 1000 information centers (Information Builders, Support Service) and has over one million users -- up 25% from its 1990's base (Paul, 1992).

In one package and with a single language, FOCUS offers an impressive range of integrated functions to include complete database management, application development, report writing, decision support, and communication. Moreover, FOCUS can be used with other database formats without conversion. FOCUS consists of a number of integrated tools and facilities. At the heart is the database surrounded by the data dictionary and security layers. The database is a multi-path, hierarchial data structure managed by inherent database management facilities. The data dictionary contains definitions of all files and database structures. Security can be protected at four different levels: at the file, segment, field, and value within a field level. FOCUS also

supports data encryption. Figure 3.3 presents the components of FOCUS.

The PC version, PC FOCUS, enables an intelligent workstation to operate as a stand-alone machine or it can communicate with the mainframe. The workstation can also operate in a client/server architecture for a true multi-user capability with the workstations linked to a database server. This arrangement offers centralized data access and concurrency control so that simultaneous access and updates of the central database can occur. FOCUS has been proven to replace COBOL for almost any type of business application. Figure 3.4 presents the savings in person-months over COBOL for a single-person 4GL project. Its weaknesses are that it can be resource intensive for high volume, on-line transaction processing and its command language is not as easy to use as other languages; but others claim these weaknesses are offset by the broad range of facilities it supports. (Martin, 1986)

FOCUS is designed for both the non-technical user as well as the applications developer through its FOCUS TALK technology. FOCUS TALK employs English-like commands, many defaults, on-line help and error correction, and a consistent syntax for its features and utilities. End users can produce reports after two days of training whereas it usually takes 14 days of training plus six months of experience to become a beginner FOCUS application developer.

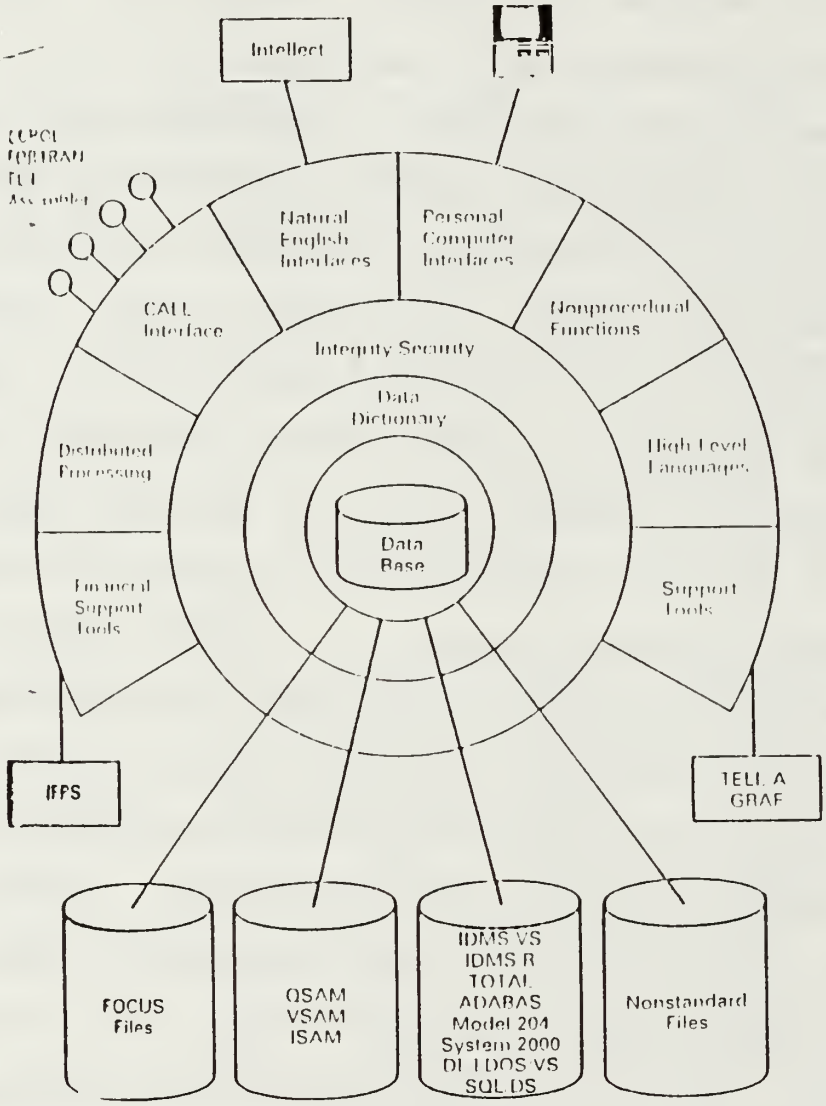


Figure 3.3 Integrated FOCUS components (Martin, 1986, p. 147)

Recall that FOCUS is a nonprocedural language -- the user does not need to know how to perform an operation, only what the operation must do. These nonprocedural requests (such as reports and queries) can be contained within procedural control statements, which dictate when and under what conditions requests can be executed, to form FOCUS executable procedures called FOCEXECS. Hence, interactive dialogue and internal testing of values are achieved.

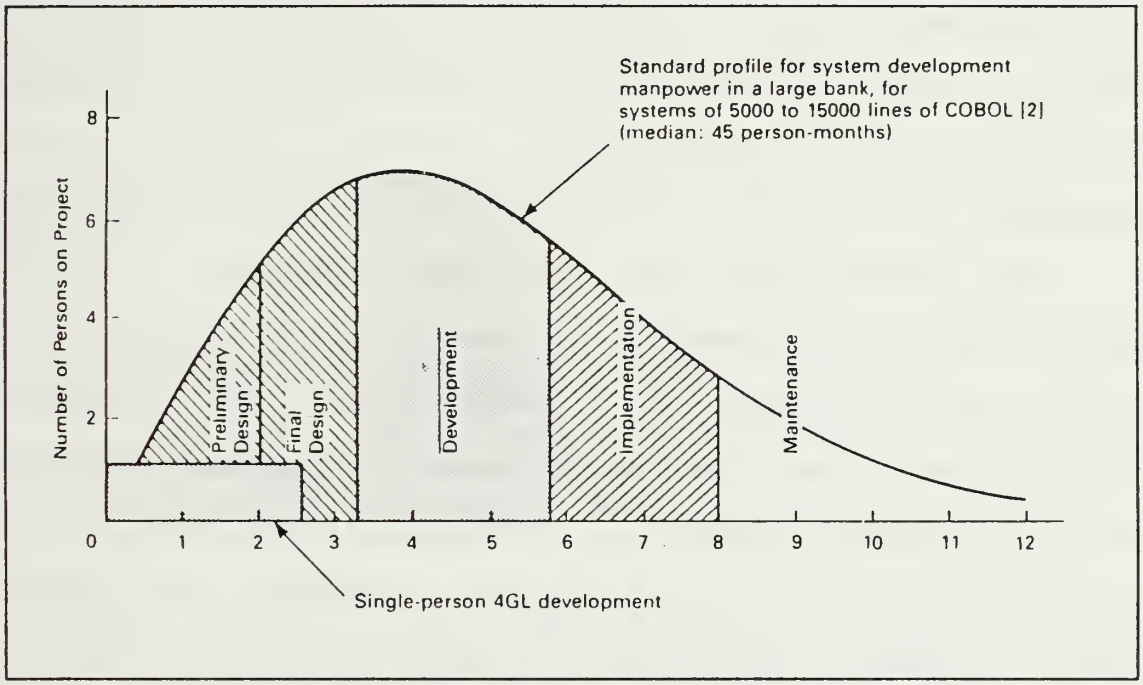


Figure 3.4 Comparison of FOCUS versus COBOL for a one-person project of moderate size (Martin, 1985, p. 81)

A group of related items of information or fields are called segments in FOCUS. For example, the personnel information for a student would be consider a segment. The collected data for one or more segments constitute a FOCUS file. For example, a student file would contain data from the

personnel and course segments. A Master File Description (MFD) is used to define the complete structure and format of the data: it contains the name of the file and the names of all its segments. For each segment, the name, format, and length of each field is defined (reference the MFD for the Minor Property Accountability System in Appendix A). The actual data itself resides in another file called a data file. (Information Builders, User's Guide, Vol. I, 1990)

FOCUS has a powerful facility known as its join operation: separate files can be dynamically joined at execution time (as long as they have a common field) to create a virtual, joined structure. This structure can also be "inverted" to minimize the input/output associated with data retrieval. (Martin, 1986) With the join command, new views of data can be created to satisfy different user needs while the individual organization of the files remains simple and straight forward. Such an alternative view is shown in Figure 3.5. "This technique does not create additional overhead, nor does it involve restructuring of the database. A different database view can substantially alter the retrieval strategy for a given query request." (Martin, 1986, p. 158)

There are several environments in FOCUS that represent different functional areas each having their own specific command sets. Two of them, TABLE for reports and MODIFY for updating data are discussed along with other features of FOCUS.

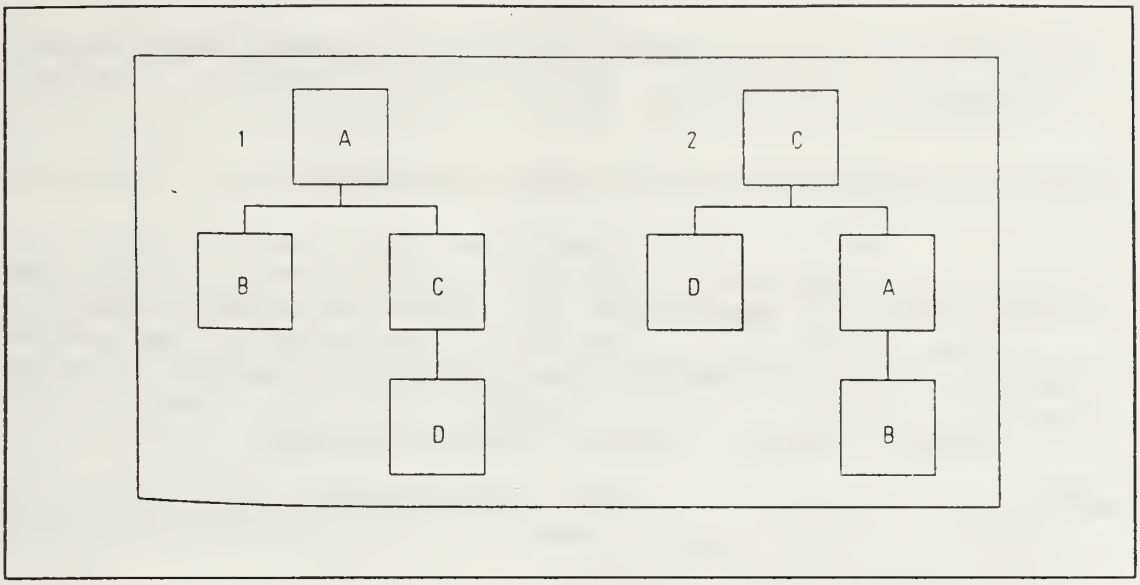


Figure 3.5 Alternative Views with FOCUS (IBI, User's Manual, Vol. 1, p. 2-227)

Features for end users include (IBI, User's Guide, Vol. I, 1990):

- **The Report Writer: TABLE** -- the files can be FOCUS files, a collection of files through JOIN, or an external file; the user can select records, perform calculations, define special fields and create custom report formats; can report on data from more than one file and has special handling of records with missing data fields; report requirements can be saved in a FOCEXEC

- **Text Editor: TED** -- has special features beyond the typical system editor; when encountering an error, puts the cursor on the error line; has split screen facilities such that four files can be displayed simultaneously; does not edit data

- **Row-Oriented Financial Report FRL (FOCUS Financial Reporting Language)** -- spreadsheet layouts, performs calculations and can carry totals forward for other reports, produces financial statements

- **Data Export Interface and File Transfer**--to prepare or format output for other products

- **The Graph Generator: GRAPH** -- the user can specify grouping and sorting characteristics and control the

format of the graph: forms include connected point plots, histograms, bar charts, pie charts and scatter diagrams

Features for application developers include:

- **Database Management: MODIFY** -- facilities for file and record handling facilities and validation and calculation features; activities include collecting data, performing validation tests, matching data against existing records, record updates, and logging of file maintenance activities; used to create MODIFY requests
- **Dialogue Manager** -- provides control facilities for creating MODIFY procedures that can include variable fields and prompts for data
- **Full-Screen Data Entry Forms: FIDEL** -- can use free-form text layout with windows and scrolling features; can protect fields and define other dynamic attributes such as highlighting, blinking, etc.
- **Database Editor: FSCAN** -- file maintenance utility to edit FOCUS databases directly on the screen; for minor corrections and changes; can scroll records, locate specific fields, add and delete records, etc.
- **Database Security** -- access rights which can vary from user to user and can be assigned to a field(s) or even to values within fields; levels of protection: no access, read-only access, update-only access, write-only access, read and write access.

D. COMPUTER-AIDED SOFTWARE ENGINEERING (CASE)

CASE or computer-aided software engineering is the automation of automation systems. It is an enabling technology for software development. CASE has evolved from a simple tool for one phase of software development to a total systems approach for the analysis, design, production, and maintenance of software. For some organizations, it represents a shift from an informal, labor intensive and

largely undocumented software process to a formalized, computer-assisted paradigm. It was founded in the concepts of structured development techniques, but unlike computer-aided engineering it was not implemented with tried and proven practices. (Pressman, 1992) It is a term that has been the subject of debate, confusion, and disappointment. Is it the answer to increasing productivity or "just another fad, more the product of vendor hyperbole than reality?" (Burke, 1991, p. 31)

The requirements for a CASE tool reflect the problems it was designed to solve: to improve productivity, to improve software quality, to improve managerial control, while being flexible and easy to use. Surely, any system or tool must be less of a burden than the problem it is trying to solve. (CASE, 1986) Some other benefits claimed by CASE include (Manley, 1990):

- Potentially speeding up the software development process
- Reduced software costs
- Automated software development and maintenance
- Automated generation of software documentation
- Automated generation of code
- Automated error checking
- Automated project management
- Formalized and standardized software documentation
- Potentially greater control of the software development process

- Integrated tools and methodologies of software engineering
- Software reusability
- Improved software portability

The essential elements of CASE include (1) procedures -- a disciplined, life cycle methodology for the development of software (2) methods -- standard design techniques and procedures for producing project deliverables and (3) integrated automated tools for:

- estimating and planning projects
- tracking project progress
- creating and modifying project deliverables
- managing design information
- reusing design and code modules
- analyzing and verifying design
- reviewing deliverables for quality
- tracing system requirements through system requirements

System analysis and design should lead to code generation with the entire process maintained by an automated methodology that can guide the developer and enforce rigor. CASE tools aid in problem partitioning, maintain a hierarchy of information about the system, produce diagrams, and apply heuristics to the specifications. (Pressman, 1992) The objective of CASE tools is to assist in application development; the programmer can ignore the specifics of coding and concentrate on the

Code generation: Tool can generate some programming language code from analysis and design representations.

Configuration Management: Tool maintains histories of document versions and configurations of documents.

Design: Tool depicts the module structure of a program being designed either in text or graphically in structure charts or modular block diagrams.

Documentation Support: Tools that provide for the extraction and formatting of the contents of the project database. Others provide standard reports, report generators, and templates to meet certain standards.

Performance Analysis: Tools that measure the complexity of software, generates static or dynamic statistics of a program's performance or analyzes the structure of a program.

Project Management: Tool provides or reports project management information including number of processes, allocation of work, completion status, and in some cases, schedules, budgets, and project dependencies.

Prototyping: Tool provides ability to develop screen or report prototypes and generate appropriate code, or provides capabilitty to rapidly develop algorithms and test the code.

Requirements: Tools providing either text or graphic capability to generate or analyze requirements.

Reverse Engineering: Tool is capable of reading source code or database schema and create the documentation and design representations necessary for enhancing and maintaining the code at the analysis and design level.

Simulation: Same as prototyping except it simulates the behavior of the prototyped system.

Strategic Planning: Tool is capable of creating an enterprise model or a strategic systems plan.

Testing: Tool provides the capbility to generate test beds or test suites from the source code. Also includes capability to assist in system integration testing in the target hardware environment.

Traceability of Requirements: Tool can track and report the impact of changes between documents or trace the development of a requirement throughout the system so compliance and completeness checks are possible.

Figure 3.6 CASE Tool Definitions (Manley, 1990, p. 22)

logic of programming versus the "housekeeping." Some general classes of tools and their descriptions are contained in Figure 3.6.

This thesis will focus on integrated CASE tools -- CASE technology that supports the entire software development life cycle (from cradle to grave) versus tools that support either

the "front end" (planning, analysis, and logical design) or "back end" (physical design and construction) of software development. I-CASE tools concentrate on the analysis and design phases of application development. The output from one high-level tool is used by another tool or is used to generate the application. As such, I-CASE tools advocate a top-down strategy to systems development. Limitations of integrated CASE include they cannot address every type of application, cannot build systems for every type of hardware platform, and cannot use every type of database, among other limitations.

To optimize functionality, the entire developmental process must be integrated through the use of automated tools. According to Pressman (1992) the benefits of integrated CASE (I-CASE) include (1) the smooth interchange of information from one method to another and from one step to the other (2) a reduction in the effort to perform the umbrella activities such as documentation, production, quality assurance, etc. (3) an increase in project control through better planning, monitoring and communication and (4) improved communication between staff members working on the same project. "Integration demands consistent representations of software engineering information, standardized interfaces between tools, a homogenous programmer interface with the tool, and an effective approach such that I-CASE can move among various hardware platforms and operating systems." (Pressman, 1992, p. 739)

The I-CASE environment combines integration mechanisms for data, tools and the user-computer interface. Ideally, information should be available to each tool that requires it; the user interface should have a common look and feel; and there should be a standardized developmental approach or philosophy.

Data and tool integration is achieved by:

- data exchange -- the ability to transfer information between <u>different</u> tools
- common tool access -- the developer can invoke a number of tools in a similar manner (through pull-down menus, windows, etc.) and can compare different representations
- common data management -- uses a single logical database of information; ensures proper check-in/out procedures; access rights and version management; includes a data merge facility and cross-project checking
- data sharing -- can use another tools' data without translation; usually a one vendor product

From a larger perspective, the information framework that supports the transfer of information between tools, data, and the user is represented in Figure 3.7. Its components consist of the shared repository (to share the data), an object management layer (to control the changes to the repository), and a tools control mechanism (to coordinate the CASE tools), and a user interface.

CASE is usually built around a project dictionary, repository or encyclopedia that stores all the information of the system and is implemented or accessed by some of the CASE

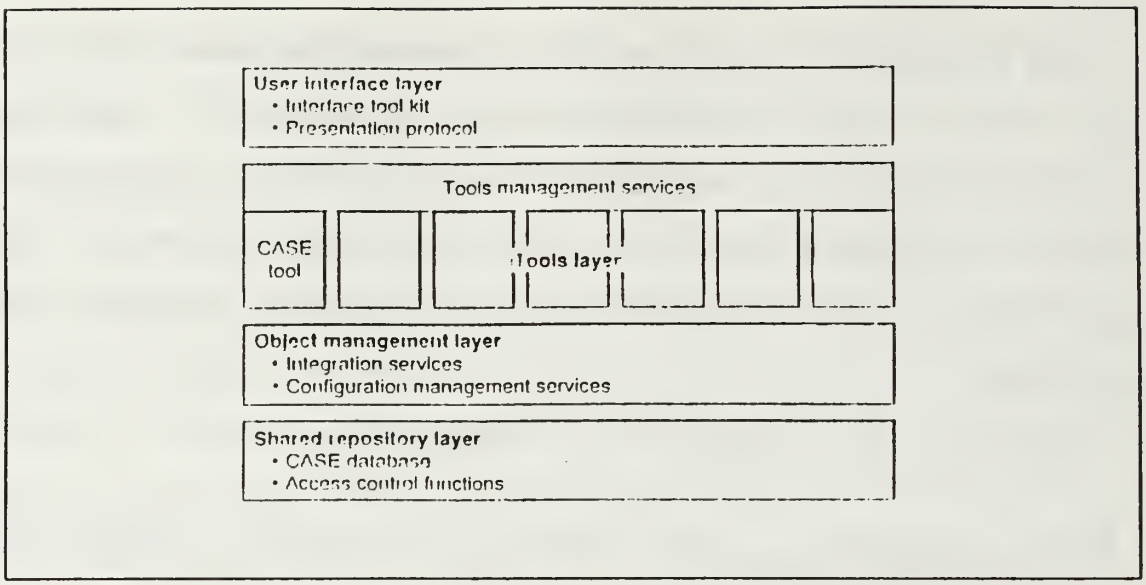


Figure 3.7 Architectural Model for an Integrated CASE Tool Framework (Pressman, 1992, p. 745)

tools previously listed. The repository of I-CASE is usually a relational or object-oriented database that achieves data-to-data and data-to-tool integration. It accumulates and maintains all application information as well as providing communication between the tools. It performs the functions of (Pressman, 1992):

- **data integrity** -- validates entities, ensures consistency, automatically performs "cascading" changes
- **information sharing** -- between multiple tools and developers, provides multi-user access, and locks and unlocks objects
- **data-tool integration** -- establishes a data model for access by all tools, controls access, and performs configuration management
- **data-data integration** -- a database management system that relates objects
- **methodology enforcement** -- a set of steps to build the contents of the repository

- **documentation standardization** -- for example, definitions from the objects are standardized

CASE tools cannot be separated from the organizational issues. The ability of an organization to absorb CASE and the associated methodology can mean the difference between success and failure. The size of the organization, the experience of the staff, and the compatibility of the methodology with the organizational environment are several factors to consider when introducing CASE into an organization. CASE can potentially cause more harm than good. "CASE improperly employed will only enable you to build more quickly the same lousy systems as before." (Burke, 1991)

From the programmers' perspective, they can no longer fiddle with the code because all changes are made to the model. To some, programming no longer seems to be an art but a disciplined engineering approach. McIninch (1992) stresses that education as to the benefits to the organization and to the staff of introducing CASE must be stated before actual training in order to increase the commitment to CASE. It is critical that the staff know not only how to use the tools, but to relate the functionality of the tool to the achievement of specific tasks. Successful implementation of CASE, therefore, requires excellent planning through anticipation of problems because every aspect -- the people, methodologies, tools, and processes are all transformed with the adoption of CASE.

According to **Datamation** dated July 1989, the most common reasons for the failure of CASE implementation are a staff that does not understand methodologies, inadequate staff training, and using CASE without management support (Loh, 1989). Petersen (1991) lists some lessons learned and recommendations that can be used for implementing CASE successively:

1. Set time boundaries to avoid over-analysis.
2. Avoid large project teams.
3. Don't let the tools dictate the deliverables.
4. The analysts have to know the business.
5. The greatest demand is for training; don't underestimate it. Recycle good people.
6. Know your requirements and limitations. Establish a solid strategic plan for what you are trying to accomplish.
7. Select pilot projects that are manageable and measure so you can evaluate the results.

E. INFORMATION ENGINEERING FACILITY (IEF)

Texas Instrument's Information Engineering Facility (IEF) is an integrated CASE tool that implements the information engineering methodology. From its initial prototype in 1984 to its first commercial release in 1987, IEF has captured 22% of the I-CASE market share worldwide and 40% of the I-CASE market share in North America as of 1989. In 1991, Texas Instruments (TI) had 300 accounts. (Penrod, 1992)

Its competitors include Anderson Consulting (Foundation), KnowledgeWare (IEW/ADW), CGI Systems (PacBase) and Intersolv (Excelerator). In a 1991 user poll of IEF, Anderson Consulting, KnowledgeWare and CGI System, IEF placed first in twelve of the nineteen categories with its highest ratings in integration of the life cycle stages, the ability to increase quality, and its code generation capabilities. The lowest ratings were received in its ability to work with other vendors' tools, support for local area networks, and the time required for training. (Sullivan-Trainor, 1991) Another benchmark test involving the integration of CASE tools and fourth generation languages was completed whereby IEF was judged against Oracle, Sapiens and other languages according to specific criteria for development and maintenance of a costing system. IEF scored excellent in speed of maintenance and integration of tools, but only fair in speed of development. (Computerworld, 1992)

TI primarily targets mainframe environments and separates its planning, analysis and design toolsets from its construction toolsets. Different construction toolsets can be purchased to match a multi-vendor, multi-platform environment. IEF supports C and COBOL code generation and PC based development for mainframe applications. Target environments include IBM, Digital Equipment and Fujitsu with DEC, VMS, Tandem and UNIX platforms recently released or undergoing testing. IEF also allows a product interface to IBM's

Repository Encyclopedia and has a Public Interface, a general purpose interface for other CASE tools and report writers. IEF is available for OS/2 on a workstation and the MVS host.

The planning, analysis and design toolsets are also available for MS-DOS. Requirements include an IBM-compatible 80286 system supporting DOS 3.1, 640K with a recommended 64KB expanded memory, and a 20MB hard disk. Cross generation from OS/2 to VMS and UNIX remote targets were released in 1991. Workstation toolsets range in price from $9400 to $23,800 and mainframe toolsets from $100,000 to $340,000. It is worth noting that TI uses IEF and uses IEF to develop IEF. (Datapro, 1991)

Version 5.0 of IEF available December 1991 includes some of the following key features: a hypertext help facility, redesigned documentation, a graphical user interface, and intelligent regeneration which determines which code modules will be affected by changes to a model and automatically regenerates those modules if required. (Texas Instruments, 1991, "Introducing IEF 5.0") IEF also supports other popular interfaces such as Microsoft Windows and MOTIF. TI has already shipped a Rapid Developer Starter Kit for $10,000 which includes the analysis, design, and construction toolsets and a tutorial, but does not include the planning toolset or full documentation.

Future trends for IEF include an ADA code generator in 1992 and a project management component as well as marketing

reengineering services and reusable IEF templates[8] for specific applications. The project management tool generates a work breakdown structure task list and an estimation of effort for a high-level data and activity model. It also creates estimates based on the system analysis, staff experience, project size, and user involvement and supports what-if analyses.

IEF components include the underlying information engineering methodology, the central encyclopedia or data repository; and toolsets for the planning, analysis, design, and construction phases of the IE software development life cycle. Normal use of IEF involves using the workstation for upper CASE activities and the mainframe for storing/receiving centralized repository information or for generating code as presented in Figure 3.8. Development stages are Information Strategy Planning (ISP), Business Area Analysis (BAA), Business System Design (BSD), Technical Design (TD), and Construction (Const).

With IEF, the user selects the stage and a corresponding list of relevant tools is displayed (Reference Appendix F). The tools used with IEF are as follows:

- Organizational Hierarchy (OHD)

---

[8] TI has nine templates as of February 1992 to include a time tracking system, general ledger package, project management system, among others. The first template was a frequent flyer program by Trans World Airlines used by Canadian Plus.

- Matrix Processor (MTX)
- Matrix Definition (MDF)
- Data Modeling (DM)
- Activity Hierarchy (AHD)
- Action Diagram (PAD)
- Structure Chart (SC)
- Action Block Usage (ABU)
- Business System Definition (BSD)
- Dialog Flow (DLG)
- Screen Design (SD)
- Prototyping (PT)
- Data Structure (DSD)

Consistency checking and model reports are also available for each stage. (Elliot, 1991) The specific advantages and disadvantages of linking the methodology with the tools, more detailed descriptions of the tools, and the strengths and weaknesses of the tools themselves are discussed in Chapter V.

IEF is a method for using information as it is a set of tools. Figure 3.9 presents the percentage of the methodology and tool support used in each phase of IEF.

The central encyclopedia installed on the host provides a centralized management facility and performs concurrent, multi-level project development. It contains a knowledge base of enterprise information such as the business' goals, analysis and design requirements. The central encyclopedia

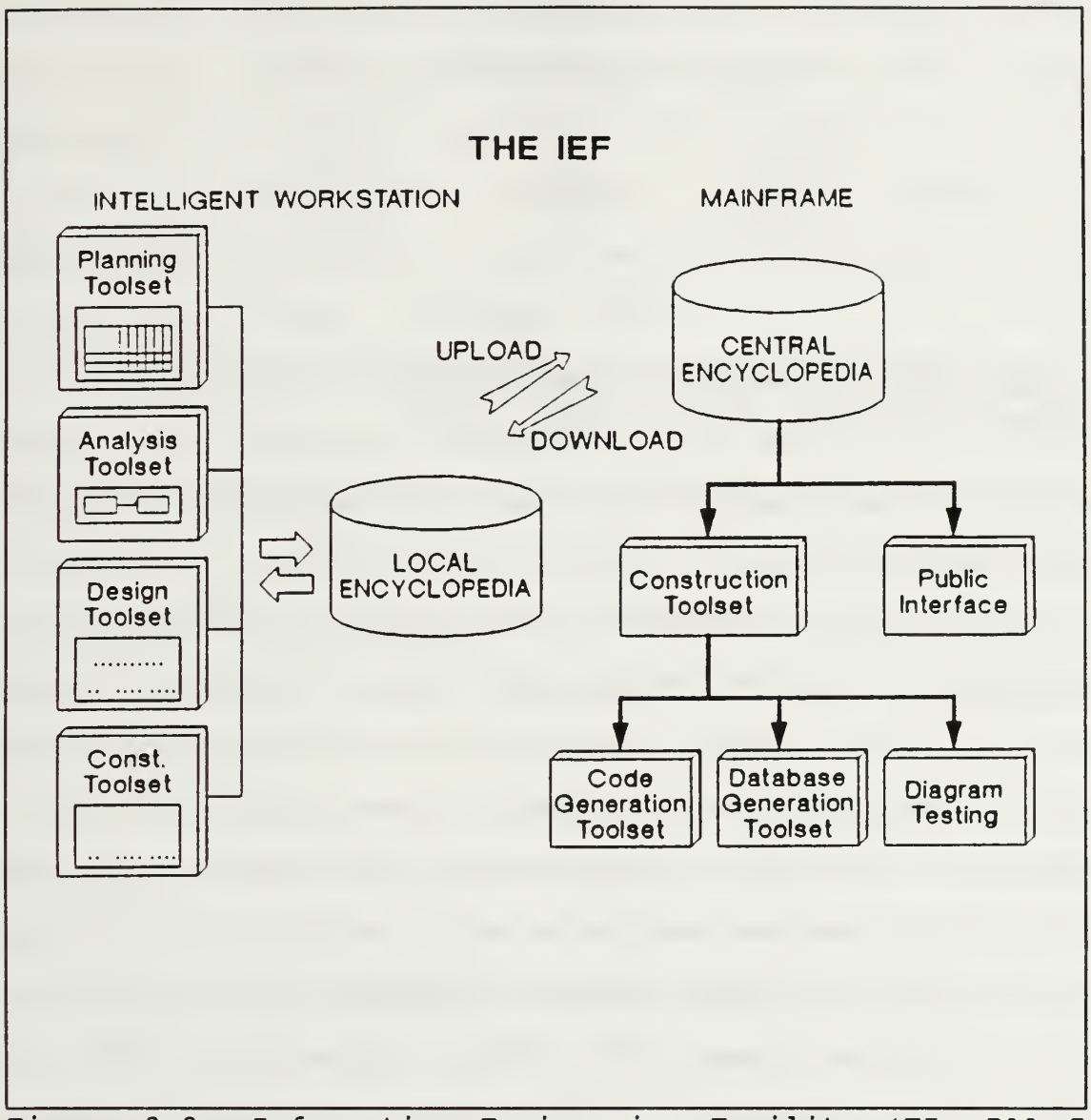


Figure 3.8 Information Engineering Facility (TI, BAA I Student Guide, Unit 1, p. 19)

supports a number of administrative features including model distribution which divides an integrated model into discrete components; model merge which allows developers to integrate modules that were developed independently into a single model; version management; and model security. Various reports about the encyclopedia can be produced including predefined reports

STAGE	METHODOLOGY	TOOL
ISP	90	10
BAA	70	30
BSD	20	80
CONST	0	100

Figure 3.9 Percentage of Methodology and Tool Support per Stage of IEF (TI, BAA I Course, 1992)

on entities, attributes, functions, user access tracking, and model contents, etc.

Integration is achieved at three levels: horizontally by maintaining integrity within each stage of development and from diagram to diagram; vertically by providing consistency from one stage of the life cycle to the other; and cross-enterprise by maintaining consistent definitions of data and activities across the enterprise and at all levels of detail. The central encyclopedia plays an important role in ensuring cross-enterprise integration through its data sharing features and reusable components. A uniform menu structure, symbols, diagrams, and terminology enables communication and integration throughout the phases. The consistent user interface is a menu-driven graphical interface across all workstation toolsets. Context sensitive explanations and help are also available for menu item commands.

The conceptual model that defines the rules and relationships directly generates the system: IEF transforms

the conceptual model during analysis into a physical model during design. During construction, IEF generates 100% of the executable code which is specific to the environment and the database management system selected. TI reports that IEF generates zero code defects as evidenced by a benchmark of 16 million lines of code. (Datapro, 1991)

Other features of IEF include rule-based consistency checking to resolve incomplete data definitions and transparent denormalization of database definitions for optimization; and applications testing. Reverse engineering, starting with code and generating a graphical diagram of the data and procedures, is not supported by IEF but is supported by TI developers.

IEF also supports what TI refers to as rapid application development within the IE framework. Once the analyst understands the system to be built, RAD can be employed to build a high quality system in a relatively short time. (TI, Rapid Development Using the IEF, 1991) As pictured in Figure 3.10, the ISP stage is omitted and if the system is new, RAD begins with the entity relationship diagram; otherwise, RAD begins with the results of the BAA. The concept of RAD with IEF is not the same as RAD as it is employed by the MIS department as explained in Chapter IV.

F. SUMMARY OF METHODOLOGIES AND TOOLS

Despite all the differences between methodologies, there are commonalities. All have three phases regardless of the application area, project size, or complexity: definition, development, and maintenance. The definition phase encompasses analyzing risks, project management (resources, costs, work schedules, etc.) and requirements analysis -- to define the information domain and functions of the application. The development phase encompasses transferring these requirements to representative code and includes testing to determine any errors in function, logic, or implementation. Maintenance is the result of change and can include correcting errors, adaptation to the software and hardware environment,

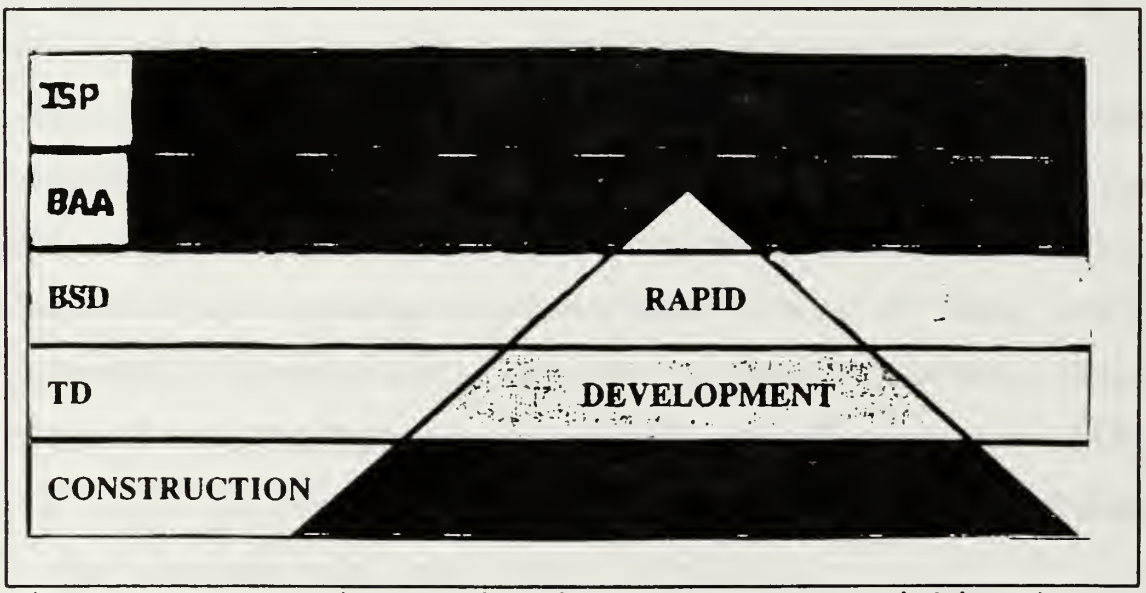


Figure 3.10 Rapid Application Development within the IE Framework (TI, Rapid Development Using the IEF, 1991, p.4)

existing software. And "You can conduct each phase with discipline and well-defined methods, or you can muddle through them haphazardly, but you will perform them nevertheless." (Pressman, 1992, p. 36)

The classical Waterfall approach is recommended for fully specified systems; prototyping for uncertain requirements; information engineering for relating business needs to information; and RAD for quick projects and extensive user involvement. Yet none of these methodologies need be exclusive of one other. The prototyping methodology can be included in information engineering and RAD. CASE tools can utilize prototyping, and fourth generation languages can utilize the business modelling concepts of information engineering. Of course, certain tools support certain methodologies better than others as the reader will soon discover. Some tools impose a methodology like IEF; others support a methodology like FOCUS.

Remember, though, that tools cannot be separated from its organization and people. An organization must thoroughly understand its software development process and apply the tools at the greatest point of leverage, and must not confuse the technology of the tool with the methodology. (Manley, 1990. p. 19) A fool with a tool is still a fool.

IV. BACKGROUND INFORMATION FOR THE CASE STUDY

A. MANAGEMENT INFORMATION SYSTEMS' SOFTWARE DEVELOPMENT ENVIRONMENT

The Management Information Systems (MIS) department has an application development group consisting of two GS-12, two GS-11 and one GS-9/11 programmer/analysts (federal job classification 334). Current applications support the Registrar, the Admissions Office, the Comptroller, and to a lesser extent the Supply department of the Naval Postgraduate School (NPS). Future possibilities for applications include integrating the curricular offices, travel and training, departmental accounting and budgeting, among others.

The MIS department has used for some time, FOCUS, a fourth generation language, by terminal emulation to an Amdahl 5995-700A mainframe located in the School's Computer Center. Each programmer/analyst receives a formal orientation to FOCUS through a series of training courses offered by the vendor, Information Builders, Inc. The application development staff serve as both the programmer and the analyst for development and maintenance. The quantity and complexity of the tasks, as well as the degree of supervision differentiate the grade levels of the staff. A typical or average system is a multi-user, single department application and will generally be

subjected to a continuing series of improvements and enhancements after its introduction.

The environment or user community the application developers serve represents an organization with an academic department component, a military and civilian administrative component, and a military student base. For example, base operations (supply, public works, and financial and personnel resources) are directed by military personnel with predominantly civilian staffs. The academic component of the School is predominantly a civilian faculty with military students and curricular officers. The challenge, for the MIS staff is to develop systems that will, more often than not, outlive those in charge, and integrate the requirements.

It has been difficult to gain support for integrated, multi-departmental, multi-user systems. The academic departments, for example, exhibit a tendency to act autonomously -- they would prefer to develop their own systems and often have the technology, funding, and labor to do so. Many departments do not have a thorough understanding of other departments' requirements or processes even though they use the same data and perform very similar functions. For example, the Research Administration department performs the same functions as the Comptroller for research funds. There is no political incentive to integrate.

Some military directors are only interested in the short-term versus the long term future of information systems since

their tours are typically only two to three years. Continued support, commitment, and resources for projects also vary with the turnover of senior managerial personnel. It is a source of frustration for the application developers to design systems that are only half-adopted by departments or are not supported by the requesting department. The Minor Property Accountability System, for example, is only used by some of the departments at the School. The Property Management Branch is not promoting it or using it to its fullest potential. MIS can provide the tools, but not the authority to comply. (Harr, 30 Sept. 1992)

The software development methodology that currently exists, rapid application development using data-driven prototyping, evolved as a natural consequence of the School's environment and represents a realistic (versus textbook) methodology for developing applications. Many of the software engineering environment factors such as extensive project management, metrics, and formal procedures present in large organizations have little utility for such a small application development staff. Metrics, as formally defined in the software engineering field, are not applied to assess the quality of the system. Additionally, there is little impetus to trace original specifications to resulting code, since those requirements are expected to change with time and familiarity with the system.

The most important component of MIS' methodology is employed before the application is started. Many projects submitted to MIS are simply the "wrong things to do", reflect distinctly parochial points of view, or just ignore the organizational and administrative infrastructure that would be required to support them. (Spencer, 27 June 1992)

The strategy used minimizes the risk of developing systems that are impossible to maintain and operate, and reflects the general difficulty that NPS managers have in coping with issues that cross the military/civilian and academic/administrative boundaries. Strassmann (1992) defines these risks as Risks of the First Kind: the risks of failures from ill-chosen goals, and Risks of the Second Kind: the ill-founded belief that actions will probably achieve the goals. These risks can be refined to include "the effects of crucial variables like the likelihood that the requirements can be specified, the probability that the specification will define an acceptable product, the frequency of reorganization, the quickness of organizational change, and the rapidity of the organization to anticipate and accommodate change." (Strassmann, 1992, p. 37)

Projects are, therefore, reviewed by MIS based on the manageable proportion of risk involved so that MIS' resources are sufficient to handle those risks and on the likelihood that a usable system will result from the investment of time and resources. Once a project is reviewed and properly

scoped, a functional manager from the respective department must assume responsibility as a pre-condition to acceptance by MIS. This functional manager is the steward of the data and to a large extent determines the application's success. MIS developers complete detail work with the functional manager and his/her employees -- the relationship resembling that of a lawyer and a client. Unfortunately, with few exceptions, functional manager are often not interested enough, too busy, or do not have the detailed knowledge necessary to define the processes of the department; and the employees that do know the details have a very narrow view of larger, integrated processes. Nevertheless, after the first introductory meeting, the MIS developers document the purpose of the application, the processes involved (why they are being done and how they relate to the subject matter), and the relationship of the functional manager to other managers with respect to the project being developed.

To comprehensively document specific system requirements early in the project development phase, the following quote from McCracken and Jackson (1986, p. 24) is offered:

> Systems requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't know them in advance--not even in principle. To assert otherwise is to ignore the fact that the development process itself changes the user's perceptions of what is possible, increases his insights into his own environment, and indeed often changes that environment itself. We suggest an analogy with the Heisenbert Uncertainty Principle: any system development activity inevitably

changes the environment out of which the need for the system arose. System development methodology must take into account that the user, and his needs and environment, change during the process.

Application developers from MIS start projects with a brief and generalized statement of the functions required. It is expected that these requirements will inevitably change and that the users' information and perspectives represent a gross approximation of what is really involved.

Because of this changing environment and the type of applications being developed -- interactive, on-line, managerial systems -- the developers follow a data-driven prototyping methodology. The data-driven methodology is based on the concept that while procedures and specific requirements may change, data is fairly consistent and stable. Alavi (1991) claims that designers using data modeling as a preliminary step to prototyping require fewer iterations, and design more efficient systems than prototyping alone because data modeling can add structure to the task. If data modeling is done correctly, there is less reliance on the accuracy of user information. Moreover, a data-driven prototyping approach that focuses on the data **instead** of the organization will lend itself to future integration regardless of the organizational structure.

Developers proceed to investigate data and their relationships by examining current reports and by interviewing

users. This step is crucial to the development of a proposed system. It is often difficult, however, to find the right person to answer the questions. Sometimes, a mmeeting with the director, a supervisor, and a clerk must be established in order to define complete relationships. In general, though, for systems developed at the NPS, the data and their relationships are pretty straight forward. Even if the system requirement information is in error, it is in a form that is relatively easy to correct.

Additional information is collected to create a working prototype. The objective is to design a working system early in the process that is "good enough." As a minimum, the end-user must use it productively. It must also meet the requirements of satisfying any integration issues involved and the needs of other functional managers that require the data. Often compromises are made based on the perceived utility of the data and to whom and for what purpose it is designed to serve.

MIS application developers use prototyping in order to determine if they are on the "right track", to determine unanticipated implications and interactions of the design, and to show management and users alike that they are working on the system (Harr, 30 Sept 1992). The prototypes used conform to the definition of an evolving prototype: they are actual working systems that will evolve into the final product, they are developed and evaluated early in the life cycle, they

provide a physical representation of the key parts of the system, and they perform a subset of the functional requirements for the entire system. There is no valid reason to create throwaway prototypes in order to evaluate a system's performance or to address proposed solutions because the hardware platforms are more than adequate, and projects are analyzed before acceptance for feasibility and commitment of resources.

The prototype usually consists of add, change, and delete functions and sample reports. The prototype is also tested according to each type of user -- the database administrator, a typical end-user of the department who frequently uses the application, and end-users that use the application in other departments or who have read-only access.

Disadvantages associated with prototyping can be avoided through project controls and experience. A large proportion of time and effort is dedicated to systems analysis so that the right system is being developed and to prevent time and resources from being spent on repair later on. The data and their relationships defined in FOCUS' master file description are reviewed by the Director before prototyping. Prototypes usually are not delayed because they can be developed quickly with FOCUS. Usually only one prototype is needed before final implementation. The prototype is also validated through a representative sample of data that reflects a full range of expected possibilities. As the data model stabilizes, the

developer moves on to the next phase of prototyping which consists of providing suitable update mechanisms and output reports of various types.

An application however is never finished. Enhancements, additional functions, or reports can be added or deleted, or new procedures may require a different menu system -- the changes are many and varied. Developers continue to improve a system until it reaches a point that requires it to be replaced or redesigned.

B. MINOR PROPERTY ACCOUNTABILITY SYSTEM REQUIREMENTS

The Minor Property Management Branch works directly for the Supply Officer as part of the Military Operations Directorate. Their staff consists of one supervisor plus three assistants who manage plant and minor property, excess property and recyclable paper pick-up. Control over minor property is required by the Comptroller of the Navy and the Naval Supply Systems Command and regulated by NAVCOMPT Manual, Vol. 3, Chapter 6. One of the missions of the Property Management Branch is to ensure adequate internal controls to safeguard and account for minor property assets. The NAVCOMPT Manual also requires each activity to formally assign a network of responsible officers, managers and custodians of minor property; to maintain a database of minor property by responsible minor property officer; and to conduct a physical inventory on a triennial basis, among other responsibilities.

Procedures for the acquisition, control, accountability, and disposal of minor property and the responsibilities of the parties involved for the Naval Postgraduate School are contained in NAVPGSCOLINST 11016.3C dated April 1992.

Minor Property is defined as property acquired for immediate use with a cost of less than $5000 or more than $5000 if the useful life is less than two years. As a minimum, all office equipment, furniture, software, etc. costing $300 to $5000 exclusive; all equipment classified or sensitive, regardless of cost, and all equipment that is pilferable with a cost of a least $100 but less than $5000 is considered minor property.. (NAVPGSCOL, 1992)

In 1990, the NPS Supply Officer had an urgent requirement to develop a database to more adequately manage minor property in light of the upcoming, mandatory FY 91 triennial inventory. The previous database was an undocumented, "home grown" dBase III Plus program. The size of the database and the fact that it was a single user, stand-alone system rendered it impractical for continued use. As of September 1990, all transactions were entered at one terminal with an approximate 30 - 45 day backlog.

One of the objectives of the MPA system is to allow the **departmental property custodians** to enter and correct entries to their own minor property: a multi-user system with restricted access and procedural controls. In addition, the system had to incorporate the approximately 15,000 records

previously entered and to produce standard reports. (Boyd, 1990) MIS accepted the project and decided to implement it as a FOCUS program on the mainframe versus a specific network since all users in the various departments could access the mainframe. The database manager for the MPA system is the Property Management Branch who is responsible for maintaining the database and related programs. Technical support for the MPA FOCUS system is provided by the MIS department.

The Minor Property Accountability (MPA) System represents, in part, MIS' methodology of minimizing risks. First, the primary requirement for the system was based on completing an upcoming triennial inventory. No attempt was made to integrate the system with any other system such as purchasing or plant property. The project was significantly "scoped down" in order to pass the inspection. An integrated and comprehensive MPA system which would serve the needs of all users was **not** constructed. To do so would have been too complex, would have taken too long, and would have involved too many people than the School and the Property Management staff were willing to afford.

The requirements were simplified at the request of the Supply Officer. For example, there was no automated method to identify components as belonging to a system; there was no requirement to provide a life history of a piece of property (from initial receipt to disposal); there was no requirement to provide a tracking mechanism for a piece of property from

include automated information systems numbers and other attributes for future uses of the data. From the Supply Officer's perspective, these were acceptable and realistic limitations although other users may not agree. The end result, however, was a useful, achievable, bounded, and relatively benign project that met Navy mandated requirements and was well suited to the skills and abilities of the Property Management staff to operate and maintain. It also served the purpose of helping managers account for their minor property.

Specific essential requirements for the MPA system include properly identifying minor property and producing standardized reports. Per regulation, minor property information should contain a locally assigned identification number; a description of the item by noun name and noun modifier; its model number, serial number, and manufacturer as appropriate; a quantity or item count; its location; its acquisition date, cost, and source document number; and the date of the last inventory. These requirements were implemented by NPS by identifying minor property with a tag number, by determining its location with a building and room number, and by assigning responsibility by a custodian's code. Additional fields include an adjusted cost in order to account for price increases or decreases to an internal system such as a computer; a Federal Supply Classification Code (FSC) listing to standardize categories of items; a remarks field; and an

action code to record the reason for deletion from a custodian's database, be it a minor property transfer, excess request, report of survey, or migration to plant property. A listing of the fields and their properties are contained in the Master File Description in Appendix A.

Other requirements included a count and adjusted cost total of all minor property items at the School and per department. The standard reports include:

1. A listing of all minor property sorted by department, building, room number and tag number.
2. A total count and adjusted cost total of all items for the School and by department.
3. A listing of minor property by serial number.
4. A listing of the FSC codes and their descriptions.

The Property Management Branch has complete access to the database and is responsible for the deletion and actions (transfers, excess, etc.) of minor property items which are usually confirmed with paper copies. They are also responsible for updating the departmental custodian and FSC listings and for maintaining standards for data entry.

The departmental custodians have access only to **their** data and can only add and correct their data. Passwords are assigned to limit access. Integrity checks are performed to ensure that there are no duplicate minor property tag numbers, that FSC codes and custodian's codes are correct, that mandatory fields are entered, and that only permitted values

for certain fields are entered in the correct format. An additional feature allows the user to have the FSC listing displayed and to select the FSC code from the list.

The database also contains information about the minor property custodians including their last name, first name, departmental code, custodian code, and phone number. A custodian cannot be deleted from the database until all of his/her property has been transferred.

V. EVALUATION

> It's now time to find ways to consistently and objectively evaluate a tool's utility and appropriateness." (Chikofsky, 1992, p. 19)

Listed are some opinions about fourth generation languages and CASE tools. Roger Barlton, of Strategic Resources Inc, states: "The move from 4GL tools to fully-integrated CASE has enabled us to make the transition from prototyping into full-blown development." (Eastwood, 1991, p.18)

> Fourth-generation languages (4GL) have been used by savvy developers for at least 10 years. In shops with an emphasis on joint application development (JAD) and rapid application development (RAD), 4GLs have proven to be the tool of choice. What we have begun to see in the 1990s is a coupling of Case with 4GLs. (Keys, 1991, p. 38)

> JMA (James Martin Associates) seems to be gunning for the sort of applications its target users usually build with proprietary languages such as Focus from Information Builders Inc, as well as trying to make CASE more palatable to a marketplace which doesn't really want to be told it's the latest productivity panacea. (McParland, 1991, p. 14)

A. INTRODUCTION

There are many factors that work against any effort to determine the most effective and appropriate methodology and application development tool for a particular software development environment. It is important to recognize these factors before developing an investigative approach.

There is currently no unified body of literature on software tool assessment nor any widely accepted systematic approaches to evaluating a tool's utility.

> That is not to say there are no credible and thorough tool reviews...but they have their shortcomings: They are often based on subjective information, at best. The evaluation criteria often change with each report. It is not clear that their results are repeatable. They are sometimes written by reviewers who have not built a coherent evaluation framework, and so are subject to vendor or developer influence. Comparisons published by different firms cannot themselves be compared. Many organizations choose a tool or toolset without establishing formal evaluation criteria or thoroughly examining the tools. Instead, they frequently base their decision on highly visible attributes such as documentation or look and feel, rather than on the quality and support of a specific method. The evaluation of a tool's usefulness is often intimately tied to a project's overall success. Seldom is an independent analysis performed that separates the tool's quality from the appropriateness of its use on the project or from the politics surrounding the project. (Chikofsky, 1992, p. 18)

It is not surprising that CASE software tends to become shelfware. One study showed that one year after introduction, 70% of CASE tools and techniques are never used, 25% are used by only one group, and 5% are widely used but not to capacity. (Kemerer, 1992, p. 23)

Another problem facing software developers is trying to assess the benefit of a tool with respect to claims of increased productivity and quality. First, one must know the definitions of productivity and quality. Is productivity a function of applied effort? Is quality determined by the product's usefulness? To an executive, a productive system may be one that is implemented on schedule, within budget,

that satisfies the users, and requires minimal on-going maintenance. Others claim productivity is a function of both efficiency and quality. Often quality is measured by how closely the system conforms to the user's requirements throughout the development process and after it is released; or it can be assessed according to the number of defects found in the system, which assumes there is a reliable method for **determining** the number and severity of the defects.

Quality measures should take into account program complexity, modularity, and size and should focus on the process as well as the product. Each evaluator should determine what is the precise relationship between the variable being measured and the quality of the software. Sometimes what is being measured is not quality, but rather some manifestation of quality. Methodological or procedural change may affect software quality. For example, defects or bugs in a program may, more often than not, be the result of miscommunication between the user and analyst than poor program design. Perhaps some of the techniques to achieving quality are the collection of correct, complete and unambiguous requirements (if that is even possible), the prevention of defects (instead of the removal of defects), and a productive environment that stresses reusability.

Unfortunately, there is a lack of well-understood and widely accepted metric models and standardized metric definitions especially for CASE products. Linsen 1988 states

that no other engineering discipline would apply methods or tools without prior extensive experimentation (that is accurate, repeatable, and controllable) that proves usability and usefulness. He cites the drug approval process as an example and proposes a CASE certification process. Metrics used for third generation or fourth generation languages may not apply to integrated CASE tools. Researchers have trouble agreeing on, not only what to measure, but how to evaluate the data that is collected. There is also an absence of robust measures to assess the impact of CASE tools, perhaps due to all the complicating and relevant issues involved: the tool's applicability to a particular project, the extent of its use and the skill level used in applying the tool. (Kemerer, 1988) A survey conducted by Loh (1989, p. 31) illustrates that "while companies generally agreed that using CASE tools increased productivity in all phases of the systems development cycle, proficiency in a particular tool, the size of the development project and the degree of tool integration all affect the results that can be obtained." Indeed, success may depend more on how the tools are used than on the potential of the tools themselves. In short, the technological issues cannot be separated from the organizational issues.

Nevertheless, the importance of metrics is generally recognized. As James Martin states (Information Builders,

Inc., Summer/Fall 1990):

> Putting metrics into place is an absolutely integral part of the CASE revolution. They can answer basic questions like 'What's your return on investment from using this tool?' You can't measure the project properly unless you have metrics in place.

Perhaps the introduction of CASE tools may lend itself as a test bed for measuring and understanding software engineering metrics and the software development process.

However, without a standard measure, companies cannot quantify the benefits of CASE which in turn, complicates and lengthens the process of justifying and implementing their use. Attempts by this author to quantify claims of productivity by requesting the quantitative approach, criteria, or measures used have resulted in subjective rather than objective responses. James Martin (1989, p. 146) does state that CASE tools provide a 5:1 reduction in total development time and a 10:1 reduction in maintenance at one-half the cost of design and coding as compared to a third generation language environment. **Where are the figures for comparison against a fourth generation environment?** Statements such as "Computer aided progression from high level overview diagrams or data models to executable code makes it possible to increase the productivity of the systems analyst" need to be substantiated with proof. How is the increase in productivity achieved, what are the measures, for what type of projects, in what type of environment, etc.? Researchers need comprehensive data of projects of equal complexity developed

in the same environment with CASE and without CASE. Most organizations do not have this luxury. It would seem that companies who work with a less rigorous methodology, have informal documentation, and do not adhere to strict structural design would see improvements in software quality due to the introduction of CASE, but that is not always true.

So how do companies justify their purchases? "Given that it is difficult to prove scientifically that CASE tools improve productivity and quality, their cost is being justified on the basis of creating comparative advantage, maintaining parity, or avoiding negative competitive scenarios, rather than on traditional return of investment calculations." (Forte, 1992, p. 71).

Similar efforts at evaluating methodologies or methods have also served to highlight the difficulties associated with the evaluation process itself. A technical report entitled "An Approach to Evaluating Software Methods" prepared by Teledyne Brown Engineering for the U.S. Army Communications Electronics Command is an excellent source for enumerating the difficulties of the evaluation process and suggests guidelines to avoid those difficulties. The report is summarized in the following sections.

The objective of a methodology evaluation is to develop practical and repeatable experiments using subjective methods to compare, contrast and evaluate the methodologies. Therefore, the first problem is to create well-defined and

repeatable experiments with credible results that can be validated. It is very difficult to claim a cause and effect relationship between the method in question and the resulting properties of the software system. How can one achieve precise control over all the other significant factors that may effect the development process, especially the human factors? Other factors include (Teledyne Brown Engineering, 1989, p. 4):

- the type of application domain being addressed
- the degree to which the user is involved in the development effort
- the point in time at which development takes place
- the size, capability, experience and needs of the development team and organization
- the physical environment in which the development team works
- the automated facilities available to support development
- the managerial procedures used or not used to oversee development

Previous investigations typically measure the resulting software and not the process of development. As the authors state, "establishing a database containing information on the emerging software production is quite different from attempting to record information about the process used to create that software." (Teledyne Brown Engineering, 1989, p.3) Justifications for assumptions made, the steps taken, and the rationale involved must be presented so that readers

can follow the thought process of the investigator in order to obtain a proper perspective of the evaluation.

Another problem for evaluating methodologies results from the nature of software development itself: most of the data collected is subjective versus objective and can be influenced by other factors of the software process:

- the data can be associated with the method itself
- a method's data can be influenced by its technical environment
- a method's data can be influenced by the development team
- a method's data can be influenced by the characteristics of the development team's organization

Other questions about subjective data arise:

- Is it practical to use subjective judgment in evaluations when such use is explicitly noted?
- Given careful statements about how subjective judgments are arrived, is it possible to retrace reasoning and repeat experiments to confirm the results?
- When both investigator and reviewer are fully aware of the assumptions made, together with the rationale for the steps taken, is the use of subjective data sufficient grounds for rejecting the investigator's results?

Once the data is collected, how do you determine which method is better than another or for that matter which is best? What baseline is used and how does one establish a measure to access relative weights or to determine rankings?

Determining fitness also complicates method evaluation since the application, user, team resources, or the

organization can, and often does, change with time. Evaluators have to decide for themselves which are the most valuable characteristics. Criteria must also be differentiated as to whether they can be determined by examining the method or by studying the development process that uses the method. For example, the fit of the method or tool may not be dependent on the development process at all but on the degree of programmer satisfaction. (Teledyne Brown Engineering, 1989)

B. INVESTIGATIVE METHODOLOGY

Given the complexities of evaluating an organization's software methodology and tool with respect to the software development environment, the author chose to approach these questions by focusing on a specific case study. The Minor Property Accountability System was chosen because it represented a bounded business area and was of limited scope and complexity. A business area is considered to be sufficiently bounded and constrained when (1) the accessed data (2) the processes including their timing and coordination (3) the business relationships with all their intricacies and (4) the business rules and policies affected by the processes and flows are all well known and clearly defined. (Haas, 1991) Given the same organization and the same requirements and scope, the Minor Property Accountability System was developed by an application developer using RAD and FOCUS, and

the author using IE and IEF. The author did **not** preview the FOCUS MPA system first.

Several approaches and techniques were used to gather the facts. Interviews were conducted with the MIS Director and application developers in order to profile the organization. Separate interviews were conducted with the application developer who designed the FOCUS MPA system and with a product specialist from Texas Instruments to answer questions about IEF. A literature review was conducted to determine appropriate measures and questions for the methodology and tool evaluations. Other organizations were interviewed to clarify concepts or features and to assess their opinions from a source other than the vendor. However, most of the evaluation was based on a "hands-on" [9] subjective evaluation of using IE and IEF by comparing it to the present methodology and tool, RAD and FOCUS.

The following questions were asked during the evaluation process:

1. What is the software development environment? What is the methodology used and what are its goals? What are the requirements or critical success factors for the organization with respect to the application developers and the user community they serve? How are these factors determined/measured? Chapter IV answers these questions.

---

[9] The author completed TI's Business Area Analysis I course, the Rapid Development Using the IEF Tutorial, and completed the analysis and design for the MPA system using IEF.

2. What are the similarities and differences between the methodologies? Can they be improved? Do they need to be improved? Is the methodology followed? Does it work? Are there advantages to an automated methodology?

3. How does FOCUS and IEF support their methodologies? How are the tools similar/dissimilar with respect to the methodology they support?

4. What do the tools do? How well do they perform their function(s)? What are their advantages/disadvantages? How do they compare to each other--what are their similarities? Does the tool "fit" the organization? Are the differences significant with respect to the software development environment?

5. With respect to the case study, how do the two MPA systems compare? Does one system do more/less than the other or implement some features better/worse than the other? What are the differences from the perspective of the user?

As often as possible, specific illustrations, experiments, or data are presented to provide the facts or proof of the evaluation. It is understood that different readers will interpret the evaluation remarks differently and affix their own measures of importance depending on their application environment and experience. However, the questions asked and the investigative approach used can be applied to a similar evaluation, perhaps of two different methodologies and tools.

For this study, several items were **not** evaluated. First, information strategic planning (ISP) and construction of the system were not conducted. Since the MPA system is a bounded system, the lack of an ISP should not affect the evaluation. Second, IEF was not being evaluated on its ability to assist multi-programmer operations or inter-model communication on a

distributed, shared system. Third, some of the benefits and costs could not be measured such as maintenance productivity, training, and cost. Obviously, the time to develop the system could not be evaluated since no metrics exist for the FOCUS system and a fair benchmark could not be achieved because the author is not an experienced user of IEF. Fourth, hardware performance was not an issue. Finally, the author could not test the application to determine if the system operated correctly although all of the procedures were put through their consistency checks. Each of these items should not affect the evaluation because the results are limited to the analysis and design phases. Further research should be conducted to include all phases of the methodologies and all capabilities of the tools.

C. TOOL EVALUATION

This section evaluates IEF against FOCUS. It determines what each tool has to offer, how well it performs its functions, and whether what it does (or what it fails to do) is significant with respect to the software development environment of the MIS organization. Only analysis, which includes data modeling and activity analysis, and design which includes programming, screen design, and report design are examined. The evaluation is based on the perspective of the application developer as well as the end user.

The format for this section will include: (1) a description of how a specific task is implemented, first, in IEF and then in FOCUS followed by a discussion of what additional features are or are not provided in one or the other tool with respect to that task (2) a discussion of the benefits and costs along with an evaluation of the results with respect to its utility to the software development environment of MIS. Obviously, every feature of the tools cannot be examined in-depth, but the basic features have been evaluated such that an appropriate evaluation can be made for the purpose of the research questions involved. Although the terminology may differ between the tools, the concepts do not. The Minor Property Accountability System will be used as an example whenever appropriate. Certain scenarios as to what steps are required to make certain changes will also be analyzed to determine the respective tool's flexibility.

1. Data Modeling

The analysis phase of software development includes determining the data, their characteristics, and their relationships. Data modeling will be the term used to define these tasks.

With IEF the entity relationship diagram (ERD) represents the logical or conceptual model of the database (a model independent of the target operating environment) versus the physical model of the database. From the conceptual

model, IEF creates the physical model and can perform certain optimizations according to a targeted environment. Since this operation is performed during Technical Design, it will not be covered in this evaluation. The elements of an entity relationship diagram can include a subject area, entity types, entity subtypes, relationships, and attributes. An entity type represents a class about which data is stored -- for example, a custodian. In relational database terms, the entity types are implemented as tables. Subject areas are simply high-level abstractions of entities such as personnel. Subject areas in IEF are for documentation and organizational purposes only. They can only be named and described and have no effect on code generation.

Partitioning can divide entities into subtypes and can be fully enumerated (**must** belong to **one** of the subtypes) or not fully enumerated (some entities are subtypes, some are not). For example, a customer's nationality can be foreign or domestic as illustrated in Figure 5.1. Nationality is considered the classifying attribute, foreign and domestic the classifying values, and the partitioning is fully enumerated. An entity subtype is more restricted than an entity and has additional common attributes and relationships. Each entity subtype is implemented as a set of optional fields in a table but IEF does not allow subtypes to exist as separately connected tables. Life cycle partitioning can also be created which is a special type of partitioning that identifies the

states through which an entity can pass, for example, from research to operation to replacement. (TI, Rapid Development Using the IEF, 1991)

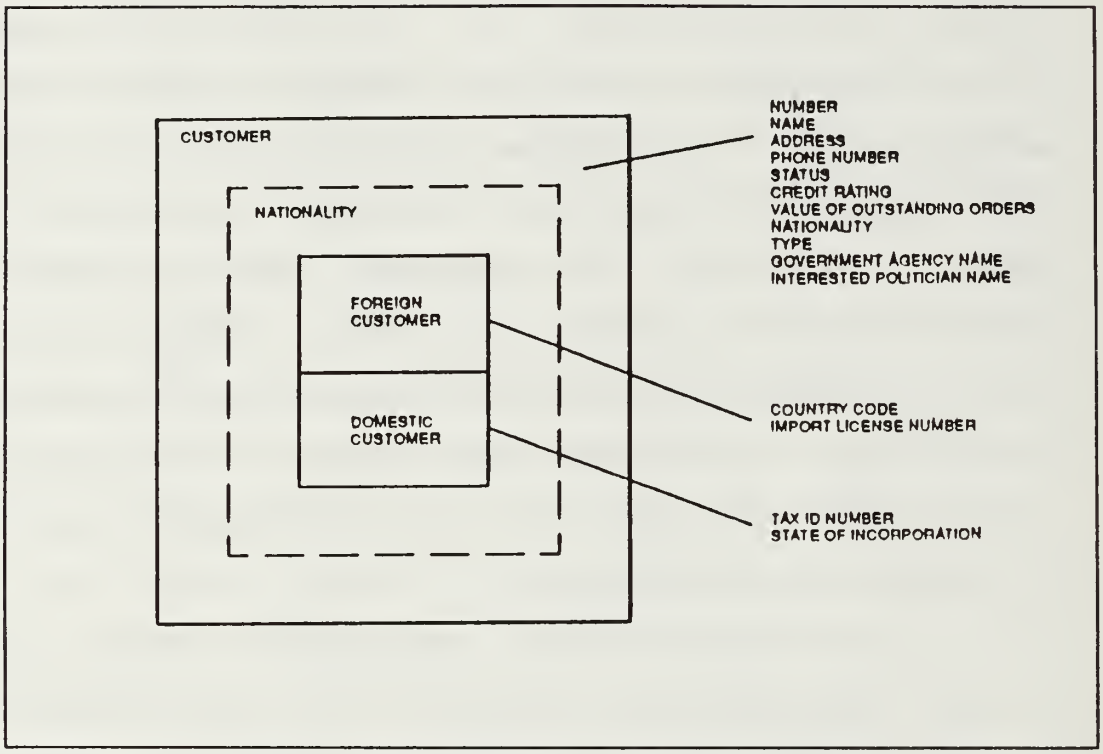


Figure 5.1 Entity Relationship diagram with Two Subtypes (TI, Guide to IEF, 1991, p. 28)

IEF captures many details of the entity types, most of which remain hidden behind the graphical representation of the entity type on the diagram. Appendix A contains the entity relationship diagram for the MPA system created by IEF. The details include its name, a text description, and its properties -- the expected number of occurrences (records) of the entity and the expected growth rate. IEF uses these volume and activity measurements to calculate data set sizes during physical database transformation. Transparent

denormalization occurs during the Technical Design phase when IEF initially optimizes the data retrieval strategy. IEF will, however, allow the developer to denormalize and change the structure and processing strategy, without having to change the logical model. It will also automatically correct any resulting update anomalies.

IEF captures many details of an entity's attributes (or fields). These include the name of the field, a text description, its optionality--whether that field is absolutely required or not, its domain (number, text, date, or time), length, number of decimal places (for numeric attributes only), case sensitivity (for text attributes only), and aliases. The source category must also be selected: either basic (the value is intrinsic to the entity and cannot be deduced from the values of others), derived (usually calculated from other values of attributes like extended price), or designed (an invented attribute to overcome some sort of business constraint or to simplify a system operation, a purchase order number for example).

IEF also allows the specification of permitted values for an attribute. The developer can specify discrete values for text attributes and may include ranges of numbers for numeric values. IEF uses these permitted values to ensure "clean" data. The values are checked when fields are entered from the screen, when data are written to a database, and when data are read from a database. For each attribute with

permitted values, a default value may be specified. IEF will assign the default value unless another value is explicitly set. For derived attributes (like extended cost), a derivation algorithm (or calculation) is specified by name only. The algorithm itself must be created as an action block using IEF's pseudocode language referred to as its action diagram.

Perhaps most important, the developer must select at least one attribute identifier (a unique key) defined as a collection of attribute(s) and/or relationships that **uniquely** identifies an entity (or record). Identifiers of entities do not imply sequence and IEF allows specification of up to five separate identifiers for each entity type. It is important to understand that identifiers cannot be modified. If an attribute that is an identifier needs to be changed, that record (and subsequently all its relationships) must be deleted and then recreated.

Relationships represent some kind of association between entities. IEF captures the name of the relationship (usually a verb), which describes the reason for joining the source and destination entities, and illustrates the relationship as a line with its associated text on the ERD. Cardinality is specified by choosing "one" or "many" (one or more). For example, property is signed for by one custodian; a custodian can sign for many property. The property of cardinality determines the placement of foreign keys, and in

the case of many to many relationships (M:N), IEF creates a link record (or associative entity). Optionality, whether the relationship must or is not required to exist, is specified by choosing "always" or "sometimes". For example, each minor property item is always signed for by one custodian but a custodian sometimes signs for many minor property items (there is the possibility that the custodian does not have any minor property).

For optional relationships, the percentage of time that at least one pairing is likely to exist can be specified. The number of pairings -- the minimum, maximum, and average -- can also be documented. For example, a custodian (95%) signs for a least 1, at most 100, and on average 50 items of property. The pairing percentage and number of pairings detail are used for documentation purposes except for M:N relationships whereby the detail is used to calculate data set size during database transformation.

A relationship can also be marked as transferable -- a pairing can be moved from one entity type to another (for example, a piece of minor property can be transferred from one custodian to another over time). A relationship must be marked as transferable or IEF will **not** allow a transfer action to occur as stated in the action diagram. Mutually exclusive relationship membership, whereby an entity can participate in one and only one relationship in the mutually exclusive group, can also be specified as a part of detailing the entity, but

is only used for documentation purposes. (TI, Rapid Development Using the IEF, 1991)

The diagramming capability in IEF exhibits some advanced options. An entity can be expanded (especially in the case of partitioning) or contracted. A locator function allows the user to select with a "rubberband" box a specific area of the entire diagram. Entities can be moved and resized with the relationships following in suit.

The consistency check of data modelling allows the user to select a particular entity or the entire ERD. Errors include "An entity type must have at least one attribute or relationship," "An entity type must have an identifier," "Derived attributes must be associated with at most one derivation algorithm," among others.

IEF allows changes to the ERD model but enforces integrity when deleting. For example, to delete an attribute from the ERD, all references to that attribute in the action diagram(s) must first be deleted. Attribute names that are changed, however, will be automatically reflected wherever they are used. Significant changes to the data model such as changing an identifier require retransformation of the model and manual procedures to interact with the database management system to update the data tables (Penrod, 1992).

Model reports include an entity definition report which provides all information about the entities (one per page); an entity definition report which provides information

about the parent entity types and their subtypes with or without the attributes listed; an attribute cross reference report which alphabetically lists the attributes' names, associated entity type or subtype, and properties alphabetically; and an attribute definition report which contains **all** information about the attributes. The Where Used Report is also helpful in identifying how a selected data object (an entity type, subtype, relationship or attribute) is being used throughout the model. Appendix A contains an example of an entity definition report. (TI, Analysis Toolset Guide, 1990)

In FOCUS, most of the information contained in the ERD diagram for IEF is contained in the master file description (Appendix A). This file can be created using FOCUS' text editor, TED, and is named in CMS with a filetype of MASTER. The actual database itself has the same filename of the master file description but with a filetype of FOCUS. Field names are equivalent to IEF's attributes; segment names are equivalent to entities; and files (or a file) are equivalent to subject areas. Entity subtypes can be created in FOCUS as unique segments whereby the additional fields for that subtype are stored in the unique segment. Life cycle partitioning is not directly implemented.

The identifiers or unique keys are defined by the segment type statement. For example, segtype=S1 means the sequence of data are logically sequenced in a low-to-high

order using the first field on the segment as the sequence key. If there was a combined key, segtype=S2 would be used. FOCUS does not allow alternative identifiers. The field type for some field names are identified as 'I' for index so that other segments from other files can link to that segment.

The field names (or attributes) can be identified by a name and forty-four character description, aliases can be defined, and the format specified (either alphanumeric, integer, decimal, packed, floating point decimal or date) with certain edit options and a specified length (except for date fields). For example, a format of D6.2 places a dollar sign in front of a six digit numeric field with two decimal places. Date formats can be YMD, Y, etc.

Field names can be "broken up" and defined for various purposes (for example, the first two characters in a date field can be defined as QTR) although the actual data for these fields are not stored -- duplication would result. The 'define' command can also be used to create temporary fields for computing new numerical values that are not in the data record. These derived attributes are simply stated as algorithms and can be defined with field names. For example, extended price can be defined as equaling unit price * quantity. The accept command is equivalent to IEF's permitted values so that incoming data values can be tested. An optional help message can be specified which will be displayed when a value fails an "accept" test, when a value causes a

format error, or when a user places the cursor in that field and uses a predefined PF help key. Edit patterns can be defined as shown by the edit pattern for phone_no in the segment custn, '999-9999'. With encrypt=on, all the fields of the segment are stored in scrambled form.

As for relationships, note the hierarchial nature of the file: the parent segments are listed first with the children below. The relationship is stated in the 'parent' statement which identifies the parent segment for that child. As presented in the MFD for the MPA system in Appendix A, each department contains one or more custodians and each custodian has one or more minor property items. Relationships between segments that are not one to many are defined as separate files such as the cross reference file or can be represented in another hierarchy and then be dynamically joined at execution, or if joined frequently, the relationship can be stated in the MFD. There is no utility to describe in text the description of the relationships or mark them as transferable, and the source and destination properties such as usage are not documented. The FSC file is defined as a static cross-reference file by its segment type statement of segtype=KU or keyed unique. The filename for the cross reference file is FSCFILE and its key is FSC_CODE. Changes made to the FSC file are reflected in all records in the minor property segment that refers to it. (IBI, User's Manual, Vol. I, 1990) In IEF, the FSC file was treated as another entity

with a one-to-one relationship with the property entity and could have been implemented as a drop-down list box. (Penrod, 1992)

A diagram of the master file description can be produced but manipulation of the diagram is limited. FOCUS does have a consistency check on the master file description highlighting the error(s) for the user. No reports beyond the master file description itself are generated; however, a program that produces a database dictionary can be purchased for FOCUS.

If changes are made to the file structure, it must be rebuilt just as the data model must be retransformed by IEF. Minor changes to the field names may not require a rebuild. Other changes such as deleting key fields or changing field formats requires an equivalent "dump and reload" operation.

Database security can be implemented in FOCUS in the master file description. Reference the MFD in Appendix A. A database administrator can be defined that has unlimited access to the file and its master file description, and must be defined to encrypt and decrypt data files. FOCUS security is provided on a file-by-file basis in which the developer specifies the names or passwords of FOCUS users granted access to a file, the type of access granted to the user, and the segments, fields, or ranges of data values restricted to the user. For example, to restrict a user with the password Clark52 to have read and write access when the value of the

dept_code is the Oceanography department (OC) the following security statement would be used: User=Clark52, ACCESS=RW, RESTRICT=VALUE, NAME=DEPT, VALUE=DEPT_CODE EQ 'OC', $. Name refers to the field the developer would like to restrict and that restriction is based on the value statement. That way, custodians can only update their own minor property records. Access attributes are read only, write only, read and write, and update only. An internal decision table that list the users and their access privileges can be displayed. The other facilities in FOCUS such as MODIFY, SCAN, and TABLE also obey security restrictions stated in the MFD.

Other security features include encryption and decryption which take place on the segment level; passwords which can be set using FOCUS by the SET PASS= command or can be set within FOCEXEC's or externally; and statistics on usage and attempted violations to FOCUS database security. (IBI, User's Manual, Vol. I, 1990)

A comparison between data modelling in IEF and FOCUS is really a comparison between the ERD and the master file description and their associated diagrams. The evaluation criteria are based on how effective the tools implement and convey the data objects, their relationships, and their characteristics (attributes or field names). It must be stated in the beginning that neither of the tools create a better data model. Quality systems analysis creates accurate data models.

Whereas the ERD represents the logical model and must be transformed in IEF, the MFD of FOCUS represents both the logical and physical data model. Both allow partitioning through subtypes in IEF and through unique segments in FOCUS, although FOCUS does not directly support life cycle partitioning. As for describing or detailing the attributes or field names, the differences are few. The developer with IEF can identify an attribute as mandatory or optional, can define a field with a time format, and select five alternative identifiers. FOCUS provides more extensive field formats for numeric fields such as floating point decimals. One of the significant differences appears in the method used to define derived attributes. In IEF, a derived attribute can be associated with an entity only if the algorithm contains the attributes of that entity. Otherwise, a separate action block must be defined for that algorithm. With FOCUS, derived fields and extractions of fields can be defined in the MFD and can include field names from other segments, and from other files if a static or dynamic cross-reference is established in the MFD.

The process of entering the data model is more time consuming in IEF as several window panels must be selected. In FOCUS, the developer can detail a field name in one to two lines. Moreover, the MFD serves as the documentation for the data model; in IEF, the same information (or less) would be printed page after page in the attribute definition report or

alphabetically in the attribute cross-reference report. In IEF, most of the detailed information remains hidden; the diagram could be greatly improved if a window similar to the MFD was displayed after selecting an entity. The MFD diagram does display the fields but is limited by the segment box. See Appendix A.

Relationships are captured differently in the two tools because FOCUS represents a hierarchial database structure versus IEF's relational database structure. Optional relationships and a cardinality of one to many is assumed for parent to child segments in FOCUS and a one-to-one relationship for cross-referenced segments. Descriptions of the relationships, their pairing percentages, and the marking of a relationship as transferable are not available in FOCUS. The latter function offers the advantage of ensuring the integrity of transfer functions when programming.

The MFD, through the detailing of the segment and through the labels on its diagram, displays the same, if not more, relationship information contained in the ERD. IEF contains more descriptive information about the relationships but most of this information is for documentation purposes. However, the diagramming capability in IEF is far superior because the user can interact with the model to move or expand entities at will whereas the MFD diagram is not interactive. From the perspective of enhanced user communication, both

diagrams would require similar training on how to interpret the diagrams.

Both tools provide a check of the data model although IEF **requires** a consistent data model before transformation. Similar procedures must be followed in IEF and FOCUS to change the data model; no automated facilities are available with IEF to reflect the changes throughout the model except when changing the name of an attribute. When adding or deleting attributes or entities, they must be deleted or added manually within the programs called FOCEXEC's in FOCUS and within the action diagrams for IEF. IEF will not allow deletion of an attribute from an entity until all occurrences of that attribute are deleted. Yet IEF does not supply a utility beyond the Where Used report to perform that function. FOCUS developers use a search and replace utility (Harr, 1992).

One distinguishing feature that FOCUS implements in the MFD is a security facility. IEF does not have a complementary, inherent security feature for its functions or data. Security in IEF is limited to the model or subset level and is implemented as a central encyclopedia function. This security feature is primarily limited to only the application development staff. Security in IEF for users must be "programmed in" to the action diagram or defined as external action blocks for use by a security tool provided by the implementing database manager. The separation of the IEF model from its data will not allow data access restrictions to

be defined in the model, as it is in the MFD with FOCUS. (Penrod, 1992) Similarly data encryption is not inherently supported by IEF.

2. Activity Analysis

To understand the functions and processes of the business and the dependencies between them, activity analysis is performed. This analysis is independent of organizational structure, existing information systems, and technology so the analyst can understand the activities of the business. The result is an activity model of the business area. Note that it is not required by IEF to have an activity model in order to do business system design, but it is highly recommended in order to understand the business rules of the enterprise and their effects on the entities involved.

It is important first to distinguish between a function, process, and procedure. A function is a group of business activities that support one aspect of the enterprise such as planning. A process is a defined business activity that represents a conceptual view of the actions required by the business. Processes have a definite beginning and end, do not generally change with time, and perform work or transform data. Processes are often related to a changed entity state such as creating an entity or significantly modifying its attributes. Note that printing reports, inquiring on available data, and recording minor changes to existing data

do not constitute processes. A process must change data, do work and produce a meaningful result to the business usually at a single point in time in one place. Procedures, on the other hand, determine how a specific action is implemented and represent the practical view. Activity is a general term which encompasses functions, processes, and procedures. (TI, BAA I Student Guide, 1992)

An analysis technique used by IEF is activity decomposition. It involves progressively breaking down business functions into smaller or lower level functions and processes. It is a form of structured outlining and is depicted in IEF by the process hierarchy diagram which can show all levels of decomposition. Appendix B contains the process hierarchy diagram for the MPA system. The goal of activity hierarchy diagramming is to identify the lowest level processes (often called elementary processes) of interest to the business with unambiguous definitions on the correct level of the hierarchy.

Building the process hierarchy diagram involves creating the root function and describing it in text (without including who, when, where or how), creating and detailing subordinate functions, and creating and detailing processes. IEF enforces the rules of activity decomposition, some while building the diagram and others during the consistency check. Some of these rules include "Functions may not be subordinate

to processes" and "The same function must not appear twice in the same decomposition."

One important activity performed while detailing a function or process is describing its expected effects. Expected effects describe at a high level how the entity type(s) may be affected by a process: the analyst can specify whether entities will be read, created, updated or deleted by using a "CRUD" matrix. These expected effects are based on the business rules. For example, a custodian is deleted from the custodian entity only after the property entity is read to determine if the custodian is still signed for property. Rules are enforced for processes as well during the consistency check.

Detailing a process involves more than describing it in text. Usage properties such as the expected frequency of execution for a process (the number of times the business expects a process to execute over a given time period) and the estimated expected growth rate (the anticipated increase or decrease in the number of executions of that process over time as a percentage per year, month, week or day) can be specified. The processes are also defined as elementary or not elementary, repetitive or not repetitive, and include a suggested mechanism for execution (batch, online, manual or other). Processes defined as elementary are added to a list of Process Action Diagrams which are built during business design. (TI, Rapid Development Using the IEF, 1991)

IEF allows changes to the activity hierarchy by allowing the developer to change a function to a process and vice versa, to change the parent of a process or function, and to delete a function or process with or without its subordinates, among other changes including inverting the hierarchy!

There are three model reports for activity decomposition: a process definition report which shows the hierarchy of activities (functions and processes with the option of including their view sets and properties such as the expected effects), the process hierarchy report which is basically a hierarchial listing of just the names of the processes and functions with or without numbers, and the Where Used report which shows how a selected data object is used by the functions and processes. (TI, Analysis Toolset Guide, 1990)

Another activity analysis technique supported by IEF is dependency analysis. Dependency is an association whereby the first activity places data (whether an attribute, sub-entity, or relationship pairing) in a certain state so the second activity can execute. The objectives of dependency analysis are to identify the sequence of processes involved, to discover missing or superfluous processes, to identify the data required to start a process and the data produced by a process, to define the type of dependency, to identify the external sources and destinations of information, to identify

events that trigger processes, and to verify the activity decomposition by ensuring the processes are at the correct level.

Dependency analysis determines the conditions needed to execute a process. The sequence is based on the dependencies as well as logic and timing constraints. There are seven types of dependencies that can be represented: sequential, parallel, mutually exclusive, repetitive, recursive and multi-enabling.

IEF diagrams the results of dependency analysis through the Process Dependency Diagram (PDD). Appendix B contains a PDD. When creating a dependency diagram, the activity's subordinates are already displayed left to right on the screen in the order they were presented in the activity hierarchy diagram. Moreover, activities added to the process dependency diagram are automatically added to the process hierarchy diagram.

External objects, such as a supplier, that are considered outside of the business area can be added and described with text. The interaction between an external object and a process does not represent a dependency but an information flow. Views can be selected which specify exactly which components of the information are required or produced by the process. Events that trigger an activity, such that the activity could not have taken place until and unless the event occurred, can also be represented on the process

dependency diagram. These events can occur at any time, such as when a shipment arrives, or can be cyclical in nature such as at the end of the year. Two different events may lead to the same activity and the same event may also trigger two different activities. IEF allows naming and describing the event with text.

The principle role of dependency analysis is to verify the activity decomposition into elementary processes. To create dependencies of any type, the activities must be joined and the dependency detailed with its name and a text description. All sibling processes should be interdependent. IEF allows changes to the dependency diagram such as changing the dependency of one process to another, transferring dependencies or events, deleting dependencies, and redrawing the diagram, among other changes. Some consistency checks are also performed on the dependency diagram. There are no process dependency reports beyond the process dependency diagram itself. (TI, Analysis Toolset, 1990)

Activity analysis is not implemented in FOCUS. Rather, it is performed by the developer during systems analysis and then documented. For the MIS developers, this documentation can contain the purpose of the system, its background history, the procedures currently implemented, the interfaces, the problem areas, and a list of recommendations. More complex or detailed requirements analysis documentation can include a description of the system and data objects, the

internal and external processes involved, and diagrams of the data flow. The developers perform activity decomposition and dependency analysis but not with a structured and diagrammatic format as in IEF. Text takes the place of diagrams and includes all processes, external entities, and events, as captured by IEF, and additional information as well.

For this specific tool evaluation, IEF's activity analysis is compared to the analysis documents prepared by the developers. In IEF, the developer can describe each of the processes but cannot obtain a collective report of the process descriptions. Each of the higher level process descriptions are simply a consolidation of the lower level processes. For simple systems, the activity hierarchy can be avoided because it resembles the structure chart.

The CRUD matrix is an effective technique for determining the effect of actions on other entities, but this matrix is only used by IEF if the action diagrams are created by expanding the expected effects. Otherwise, stereotyping as presented in the next section achieves the same results without activity analysis.

The process dependency diagram is an effective diagram to summarize the processes involved, their dependencies, and the events and external entities that trigger the processes. However, the diagram cannot stand alone. IEF does not provide enough information to completely describe the activities of a business as contained in a systems analysis document. The

developer must go from process to process to read the short descriptions in order to understand the activity being modelled. IEF should include as a minimum a text file with each dependency diagram. Moreover, the time required to create the diagrams is extensive.

This evaluation does not mean activity analysis should not be performed -- on the contrary! It only recommends that activity analysis as implemented by IEF is incomplete. The reports are difficult to read because they are basically lists of processes and their details. Systems analysis documentation which includes many of the business rules achieves the same results and is more user-friendly.

For more complex systems, activity analysis may be an effective tool for the developer to decompose the functions involved and to understand the interdependencies between the processes. The investment in time, however, should be weighed against the benefit achieved by the tool. Recall that activity analysis is not required in IEF for code generation. For simple systems, an experienced application developer can analyze a system, document it, and then proceed directly to programming.[10]

---

[10] The Rapid Development Tutorial Module did not include activity analysis. Whether it was avoided because the system being modelled was simple or because it is not part of rapid development for IEF was not satisfactorily confirmed by the author.

3. Action Diagramming or Programming

The Process Action Diagram (PAD) in IEF is considered the product of interaction analysis--determining the effects that processes have on entities, attributes, and relationships. They are IEF's high level computer programs whereby each PAD contains the detailed process logic for elementary processes, procedure steps, business algorithms and derivation algorithms on which code generation is based. Process action diagrams detail those actions needed to produce output from input. Procedures, on the other hand, implement or incorporate the process action diagrams and identify how the system interacts with the user. As an analogy, the process action diagrams are the subroutines and the procedure, the programming logic contained in a menu. Both process and procedure diagrams are built with the action diagramming tool and will be discussed.

Building an action diagram consists of three basic activities: creating views, defining the logic, and defining the action blocks. A view is a collection of associated attributes that are needed as input to or output from a process or procedure. There are four groupings of views included in every action diagram: import, export, entity and local views. Import views provide the information the procedure or action block requires to start execution and those attributes marked as mandatory (as opposed to optional) must be present. Export views provide the information that is

required at the end of execution. Entity action views provide the information which the procedure or action block inspects or manipulates during its execution. Local views are generally used to temporarily save information during execution. Repeating group views are created for multiple occurrences of entities like lists. As an example, the import view for displaying a minor property item is the property entity with only its tag number attribute (since the tag number is all that is required to retrieve the correct minor property item); the export view would be the property item with as many attributes as the developer wishes to be displayed (and can include the FSC and custodian entities and their respective attributes as well); and the entity view would include all the entities referenced in order to produce the export view. The import and export views can be automatically created in the PAD if they were defined with the activity hierarchy or dependency diagram. If not, PAD view maintenance can be used to change, copy, delete and add views to the action diagram.

The next step is to define the processing logic for the action diagram. In order to add any action statements, the views must have been created first. There are actually three methods to build process logic in PAD's: by scratch, by stereotyping, and by expanding effected effects. The last method is interactive whereby IEF recalls the expected effects for each process defined during activity analysis and guides

the developer through the available options to construct the action statements. Unlike stereotyping, no assumptions are made.

For basic processes, stereotyping is the preferred method. Stereotyping will create the views, expected effects, definition properties, and action statements automatically. The developer only has to name the action block and then choose which action is involved: create, read, update, delete or list. All the PAD statements are automatically created with the correct entity names and process logic including all necessary relationships to ensure integrity checking. The exit states such as property_nf for property not found or property_ae for property already exists are also automatically created. Appendix C presents a generated PAD for updating a minor property item. IEF performs cascade deleting unless directed otherwise: if a record is deleted all other records that participate in a mandatory pairing relationship are also deleted. The developer then adds the specific action statements or logic required to reflect the business' rules. (TI, Design Toolset Guide, 1990)

The most striking difference between IEF and other programming languages is that the developer is not allowed to type in any statements. Rather, the developer is guided through the creation of each statement by clicking on the appropriate next word or phrase. This feature is known as machine-led dialogue. In one respect, IEF presents only valid

choices such as the entities, attributes, views, and actions. The types of actions (not a complete listing of all the commands used with the actions) consist of:

- Entity actions (read, read each, create, update, delete) to retrieve and manipulate stored information about entities
- Relationship actions (associate, disassociate, transfer) to manipulate pairings of stored entities
- Assignment actions (set, move, exit state is, printer terminal is, make) to assign values to attribute views
- Repeating actions (read each, for each, while, repeat until, for) to manipulate components of a repeating group view--a list
- Conditional actions (if, case) to change the flow of the process based on some condition
- Control actions (use, escape, next) to change the flow of the process unconditionally

There are also text, number, date, and time expressions such as last, max, current date, and spaces which are used as tests in conditional statements and in set assignments. Comments can also be added with the command "NOTE." (TI, Rapid Development using the IEF, 1991)

IEF allows the developer to copy a PAD with substitution whereby an entity name can be substituted for the original entity and the action diagram will be automatically created for the new entity. However, copy with substitution only works with **one** entity type and no referenced relationships (like create custodian).

Within the PAD itself, simple changes such as changing the name of a view and complex changes such as moving, copying, and deleting action blocks can be accomplished by selecting the option from the menu. IEF will automatically highlight all the statements involved for that action block; in other words, components of an action statement cannot be inadvertently separated from its action block. Movement throughout the PAD is facilitated by menu options such as bottom, find, goto, etc. or by using the mouse. Consistency checks can also be performed on individual PAD's.

Common action blocks, those invoked by more than one process or procedure such as algorithms, are referenced in the PAD by the 'Set Using' or 'Use' action commands. External action blocks which define the interface between IEF and subroutines outside of IEF can also be created. The command 'External' informs IEF that a non-IEF-generated program will be used to obtain the data needed for that action block's export view(s).

Before creating the procedures that implement the processes, a business system must be defined. Basically, this involves naming the system and setting system defaults to achieve standardization for the entire business system. These system defaults include common commands; synonyms for those commands such as 'A' for 'Add'; function key definitions; common exit states (a message that appears after execution of a process such as "operation successful" or "display before

updating"); certain display properties for fields on the screens such as highlighting and color; and certain edit patterns for output like YY/MM/DD.

As mentioned before, procedures are created to implement one or more processes (process implementing procedures) or can be created to improve the implementation of the business system (designer-added procedures like the main menu which calls other procedures). A procedure may also consist of several procedure steps, with each step usually associated with a screen in an online system, but frequently procedures consist of only one step. (TI, Design Toolset Guide, 1991) An example of an IEF process-implementing procedure is minor property maintenance because create and update processes occur.

Like processes, each procedure is associated with an action diagram and can be automatically created through a process called transformation. The developer starts the transformation by naming a procedure and selecting the elementary process or processes to be implemented from a list of processes. This decision often depends on the needs of the users. A procedure may implement only one process for simplicity's sake, may implement one process over and over again if executed frequently, or may implement several processes. For the last implementation, the information required by the elementary processes should be very similar and the same set of users should perform all of the processes

used by the procedure. IEF continues the transformation by ensuring all the elementary processes are consistent; by requiring the developer to select the commands that will invoke the selected processes; by synthesizing the procedure action diagram by "calling" each process with the 'Case' and 'Use' statements; and by combining all the views of the elementary processes into the views for the procedure. (TI, Rapid Development Using the IEF, 1991)

Appendix C presents the procedure action diagram for Minor Property Maintenance. Note that it implements several processes based on the 'Case of Command' statement and that the views for the procedure represent the collection of the views for all the processes. Consistency checks can be executed against procedures and the import and export views will be automatically linked to the procedure's corresponding screen.

Two related tools, the Structure Chart Tool and the Action Block Usage Tool graphically display the action blocks in the model. These tools can be used to refine the processes during activity analysis and to refine the procedures during business system design. The Structure Chart Tool displays a diagram of the hierarchy of the actions blocks used by a selected process, procedure, or action block whereas the Action Block Usage Tool displays where an **individual** action block is used by a process or procedure. Appendix C presents examples of both tools.

The Structure Chart Tool is used to analyze action diagrams and to represent them graphically. It eliminates the need to chain between all the action diagrams to review the structure. "The branches of the structure chart represent the paths to lower level action blocks referenced by the USE command in the action diagram." (TI, Design Toolset Guide, 1990, pg. 18-3) The structure chart can be drawn vertically, horizontally, or indented. All or selected levels can be expanded or contracted and consistency checks can be performed on the process, procedure, or action block. Action block and view maintenance can also be performed.

The Action Block Usage Tool shows how an action block has been referenced through the USE command by other action blocks, processes, and procedure steps similar to an index reference in a book. The branches of an action block usage diagram represent the paths from each higher level action block to the **same** lower level action block. (TI, Design Toolset Guide, 1990)

In FOCUS, the system defaults are established in a file named profile which can be used to established standards, conditions, and resources during a FOCUS working session. Another profile or FOCEXEC can be established for the user to specifically run a program, set their password, etc.

In FOCUS, the general purpose text editor called TED is used to create and modify all files used by FOCUS (except the databases) and is equivalent to IEF's action diagramming

tool. TED's advanced features include moving or copying lines of data from one window to another using its split-screen facility, the ability to access the screen painter, and the ability to immediate execute a FOCEXEC from within TED.

There are four environments to TED: type, edit, input, and paint. Type provides easy-to-use commands such as add, replace, top, etc. to edit and create files. With edit, the developer can also access the prefix area, the six left-most columns on the screen, to enter commands to delete a line or move a block of text, for example. Up to four files can be edited at the same time. Input is used to enter text without predefining the amount of space and is primarily used to create new files. The paint command is used to provide access to the FOCUS Screen Painter. A FOCEXEC or executable procedure can also be edited with TED; and by issuing the 'run' command, the FOCEXEC can be executed from within TED and the error line, if any, will be highlighted (i.e. a testing facility within TED).

The MODIFY facility in FOCUS is used to maintain FOCUS databases through adding, deleting, and updating data. These requests indicate which database to modify, the method of reading the data, the search technique, and the actions required. The following request, for example, adds new employee data:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID LAST_NAME FIRST_NAME
```

```
MATCH EMP_ID
    ON MATCH REJECT
    ON NOMATCH INCLUDE
DATA
```

The request will modify the file employee; prompt the user for the employee's ID, last name, and first name; will search the database according to the employee ID entered; if the ID already exists (ON MATCH), the request is rejected; if the ID is not in the database, the request is granted; and then the user is prompted to enter more data. Similar requests can be made for updating and deleting data. Appendix C presents the FOCEXEC for updating a minor property item.

The MODIFY request can perform other tasks such as test values for accuracy using the VALIDATE command; perform calculations; modify incoming data fields; and define temporary fields with the COMPUTE command. The request can display messages using the values from the input fields with the TYPE command; modify multiple FOCUS files in one request; record execution statistics; and use CASE logic to branch to other requests, among other tasks. However, like IEF, key fields cannot be updated with MODIFY. Child segments can be modified directly, without accessing the parent segment first, as long as the seqment has one key field that is indexed. The NEXT statement can be used to modify or display data in the entire root segment, or with case logic, can display all descending segments. The LOOKUP function can be used to

retrieve data values from cross-referenced files. The COMBINE command allows modification of two or more FOCUS databases in the same modify request by combining the logical structure of the FOCUS files while leaving the physical structures untouched. FOCUS permits multiple record processing by retrieving many instances of a segment based on a key field value, such as retrieving the pay records for each month of the year for an employee. FOCUS contains a feature called absolute file integrity with its associated commands COMMIT and ROLLBACK, which safeguard the database in case of hardware or software failure by writing to the database only when the request executes properly. Valid CMS commands can also be issued from within FOCUS by using the CMS prefix.

Although the MODIFY facility is used for extensive database maintenance, FOCUS provides FSCAN (and SCAN) for making minor changes. FSCAN displays databases as if they were flat files on the screen and allows scrolling through the database, locating and changing specific values, and file editing such as adding, updating, and deleting records. To move from one segment to another, the user must first move from the parent segment to the child segment. **All** of the children for **all** parents **cannot** be displayed -- only the children of a particular parent. FSCAN also allows the developer to change key fields. (IBI, User's Manual, Vol. I, 1990)

An evaluation of the programming tools of IEF and FOCUS is based on how effectively each tool constructs programs and process logic and facilitates data maintenance. The TED and MODIFY facilities of FOCUS will be compared to process and procedure action diagramming. Recall that both processes and procedures in IEF are implemented as FOCEXEC's in FOCUS.

The text editor, TED, of FOCUS and the action diagramming tool of IEF perform similar functions in constructing programs although the action diagrammer uses machine-led dialogue. This technique is claimed to prevent errors from occurring and assists the developer in creating the action statement by presenting only valid choices. These same errors would be caught by FOCUS after the fact by "running" the FOCEXEC. IEF's machine-led dialogue may be effective for the beginning user, but having to click on every word, phrase, and option gets to be annoying in this author's opinion. If IEF can determine valid choices, why not add a text editor to FOCUS with on-line error checking? Note that IEF will still not catch bad logic with machine-led dialogue. To prevent syntax errors while programming in FOCUS, the developers simply have a copy of the MFD in front of them (Nolan, 1992). The action diagramming lines that connect the logic and action blocks in IEF do assist in readability especially if a programmer does not indent and use structured programming techniques. The longer descriptive names of the

processes also aid in identifying the purpose of the process.

The requirement to have import and export views in every process action diagram (PAD) in IEF is equivalent to creating input and export screens/reports in FOCUS. Entity and work views do not have to be specified in FOCUS since the file only has to be named in order to include all of its segments and field names, and absolute file integrity is inherently incorporated.

The justification for entering views in the PAD's stems from the concept that a process should use only the minimum information that is required for execution. By "starving the views", the developer has a definitive and complete understanding of the process involved. (Penrod, 1992)

While this concept serves its purpose, the author believes that views should be defined in the analysis phase (which is a feature of IEF) and not implemented in the PAD. Instead, view information required for entering and displaying data can be defined during screen design. Granted, IEF would lose its current **PAD**-to-screen view consistency (views stated in the PAD must match those in screen design) especially if activity analysis was not performed.

Having to add views in **both** the action diagram and then collectively in the procedure, seems like unnecessary duplication. View maintenance is not automatically generated except during initial procedure synthesis. For example, every time an attribute must be added to the screen, the attribute

must be added to the process action diagram view first. The price for eliminating views is potentially bad screen design, possibly too much information, and if analysis was not performed, an incomplete and misunderstood process. The benefit would be maintenance flexibility and more rapid application development.

The stereotyping and copy with substitution features of IEF have limited benefits. Stereotyping will save time initially to create "first-cut" action diagrams, but all other action statements added thereafter to reflect the business rules are not saved if stereotyping is re-executed. Unfortunately, the copy with substitution command will only execute when one entity is involved which for moderately complex systems, is not the case. A FOCUS developer would simply copy the FOCEXEC and use the replace command.

Notice the differences between the two high level programming languages as illustrated by comparing the Change Minor Property Item FOCEXEC with the corresponding Update Minor Property action diagram in Appendix C. Both programs obtain input values, perform integrity checking, select the correct occurrence, and update the database for that single occurrence or record. Consider, for the time being, that the screens (called crtforms) in FOCUS and the import and export views of IEF are equivalent.

Note that the relationships or segments do not have to be restated with FOCUS when performing integrity checking on

the custodian and minor property segments. The relationships and entities must be stated again with IEF even though they are already defined in the data model. For example, " MODIFY file minor, MATCH tag_no" in FOCUS is equivalent to "READ property where property tag_no is equal to import property tag_no" in IEF.

The exit states of "already exists" and "not found" in IEF are equivalent to FOCUS' "ON MATCH type" and "ON NOMATCH type" commands. The action commands of create, read, update, and list are similar to MODIFY's commands except list would be executed as a report (a TABLE request) in FOCUS. With update in FOCUS, there is no need to SET every attribute to its import attribute as implemented in IEF. In FOCUS, it is assumed that the values entered will be used to update the data because they are defined as turnaround variables on the screen.

A similar comparison can be made between the procedure action diagram of Minor Property Maintenance and the FOCEXEC that initiates the main Minor Property Maintenance Menu in FOCUS as contained in Appendix C. Both programs, based on the option selected, branch to execute the appropriate function. The most striking difference is the inclusion of all the views, not only in the beginning of the IEF procedure itself, but in each CASE action section as well. All imports must be moved to exports for each CASE section in IEF. In FOCUS, the user selects a menu option through a local variable, &opt, and

then branching locates the appropriate label and the appropriate FOCEXEC is executed. Note that the use of GOTO's can be changed to CASE logic.

There are other differences between programming in IEF and FOCUS. Certain common CMS commands in FOCUS can be included in the FOCEXEC's whereas all action statements outside of IEF must be executed as external action blocks. Because IEF is separated from the database management facility on the targeted environment, IEF does not have an equivalent FSCAN capability to browse the actual data. Granted, this facility should be used with care in order to maintain data integrity.

IEF's structure chart and action block usage diagramming tools offer distinct benefits. The structure chart would serve as an excellent documentation tool for the user and the action block usage tool as well as the Where Used Report could assist the developer in mapping program usage. FOCUS could use these utilities.

Having a long process or procedure name (greater than eight characters) and a description also aids the developer in identifying its function(s). The separation of procedures and action blocks as listed by IEF assists in organizing and understanding the system. Unfortunately, they are listed alphabetically instead of grouped according to their common entity. Currently the FOCUS developers systematically name their FOCEXEC's according to the processes they implement, but

no generated map as to which FOCEXEC's are called from within other FOCEXEC's is available. The logic of the system is instead documented by a system flow chart, matrix, and/or a menu hierarchy as presented in Appendix D.

In conclusion, the TED of FOCUS offers flexibility and immediate testing whereas the action diagramming tool of IEF prevents mistakes through machine-led dialogue. Programs in FOCUS are simpler and can include external commands and a tool to scan the data. Action diagrams in IEF contain the minimum information needed for that program through view maintenance. Ironically, this requires specifying that information to a low level of detail. The programming concepts of FOCUS and IEF are basically the same except more lines of code are required in IEF to achieve the same results in FOCUS. Management of processes and procedures is superior in IEF. For constructing programs quickly when prototyping, IEF's stereotyping feature has the advantage. For program maintenance, FOCUS has the advantage. This evaluation is incomplete, though, because the interaction of dialogue or control flow and screen design are also important to the construction of programs.

4. Dialogue Flow

When a user invokes a procedure in IEF, there is a lot of interaction among the procedures, action diagrams, screens, and dialogue flows. Dialogue flow transfers control and data from one procedure to the other and will be discussed shortly.

First, an overview of the user's interaction and an explanation as to how IEF's components work together is necessary.

First, it is important to note that the action diagrams are independent of the screen and dialogue flow. A user starts a procedure by entering data which is then mapped to the import view for that procedure. The appropriate action diagram is executed and the results are mapped to the export view for that procedure. Based on a condition set by the action diagram, the procedure can either display a screen -- the export view is mapped to the screen -- or flow to another procedure in which the export view is mapped to the import view for the next procedure. This follow-on procedure can be executed immediately or another screen can be displayed in order to obtain more data. (TI, Rapid Development Using the IEF, 1991)

A dialogue represents the interaction between a user and each procedure of the system and is implemented through the use of screens. With IEF, the dialogue flow diagram is used to represent the possible paths a user can travel through the system via screens. Appendix D presents the Dialogue Flow Diagram for the MPA system. Dialogue flow details the sequence in which procedures occur, defines the data that is passed between procedures, specifies the conditions under which control is passed between procedures, and defines the

function keys used to invoke the commands used in the procedure.

Two types of dialogue flow can occur: a transfer, in which control and optionally data is passed from one procedure to another; or a link which is the same as a transfer, except that control and data may be passed to the **source** when the destination procedure is complete. An example of a link dialogue would involve obtaining information about a customer in order to establish an order, performing a credit check, and then continuing to establish the order without any original data loss.

Each link must have a defined 'Flows on' exit state which represents the condition necessary for the flow to take place as well as a 'Returns on' exit state which causes control to be returned to the source procedure. No explicit commands are necessary for accepting and displaying screens. Commands and function keys can also be associated with dialogue flow. For example, the exit state 'Go to FSC Maintenance', the command FSC and its synonym 'F', and the function key F6 all can execute the same dialogue flow to FSC Maintenance. The developer must determine whether the procedure should be displayed first in order for the user to input more data, or executed first such as when displaying lists. The developer can also specify what data should be passed by selecting the appropriate view. Flows can also be defined between IEF and non-IEF transactions through action

diagram statements but cannot be diagrammed. Consistency checks can be performed on the dialogue flows. (TI, Rapid Development Using the IEF, 1991)

Some of the model reports generated during business system design include a command list report that lists all commands and synonyms defined for the business system; an exit states list report which lists the exit states and their corresponding messages; a procedure definition report that contains a list of selected procedures, their descriptions, their procedure steps, processes implemented, and name of the corresponding business system; the procedure step definition report which contains more information such as the procedure step's dialogue flow properties and view sets; and the where used report which can be used to determine how an exit state or other selected object is being used by the procedure steps. (TI, Design Toolset Guide, 1990) Appendix D contains an example of the procedure definition report.

IEF includes a prototyping feature that enables the developer to demonstrate online screen dialogue flow without having to build the action diagrams. This technique can verify the user's requirements. The order and the content of the screens can be reviewed but no data can be entered, interpreted, transferred, or simulated on the screen. All the procedures, screens, dialogue flows, and function keys must be defined to use prototyping. (TI, Design Toolset Guide, 1990)

Testing can be performed at the action diagram **level**

interactively but not while constructing the action diagram itself. The code must first be generated. With 'action block call trace on', the developer is presented with a list of the action blocks called and the calling sequence -- which statement number in the process or procedure was the action block called from or from which tool. Figure 5.2 presents a sample action block trace screen. The developer can watch the execution of an action diagram and step through each of the statements, inspect and modify the views, and change the screen display, as well as test data. Since testing is contained in the Construction toolset which was not purchased, the author could not evaluate testing. (TI, Rapid Development Using the IEF, 1991)

In FOCUS, the Dialogue Manager is used to build, manage, and control its executable procedures, the FOCEXEC's. All requests can be written using the TED editor in a FOCEXEC

```
                  Action Block Name                 At Stmt #

                  Customer_Maintenance              Dialog Mgr
                  Add_Customer                      0000000002
Entering ----> Verify_Customer_Credit               0000000024
```

Figure 5.2 Example of an Action Block Call Trace Screen (TI, Rapid Development Using the IEF, p. 208)

and executed by typing 'ex' followed by the filename. Dialogue Manager control statements (such as exit, goto, if, prompt, read, etc.) refine the requests and are executed first. Regular FOCUS commands are stacked and are executed only by the -run statement. Figure 5.3 presents the sequence of FOCEXEC processing.

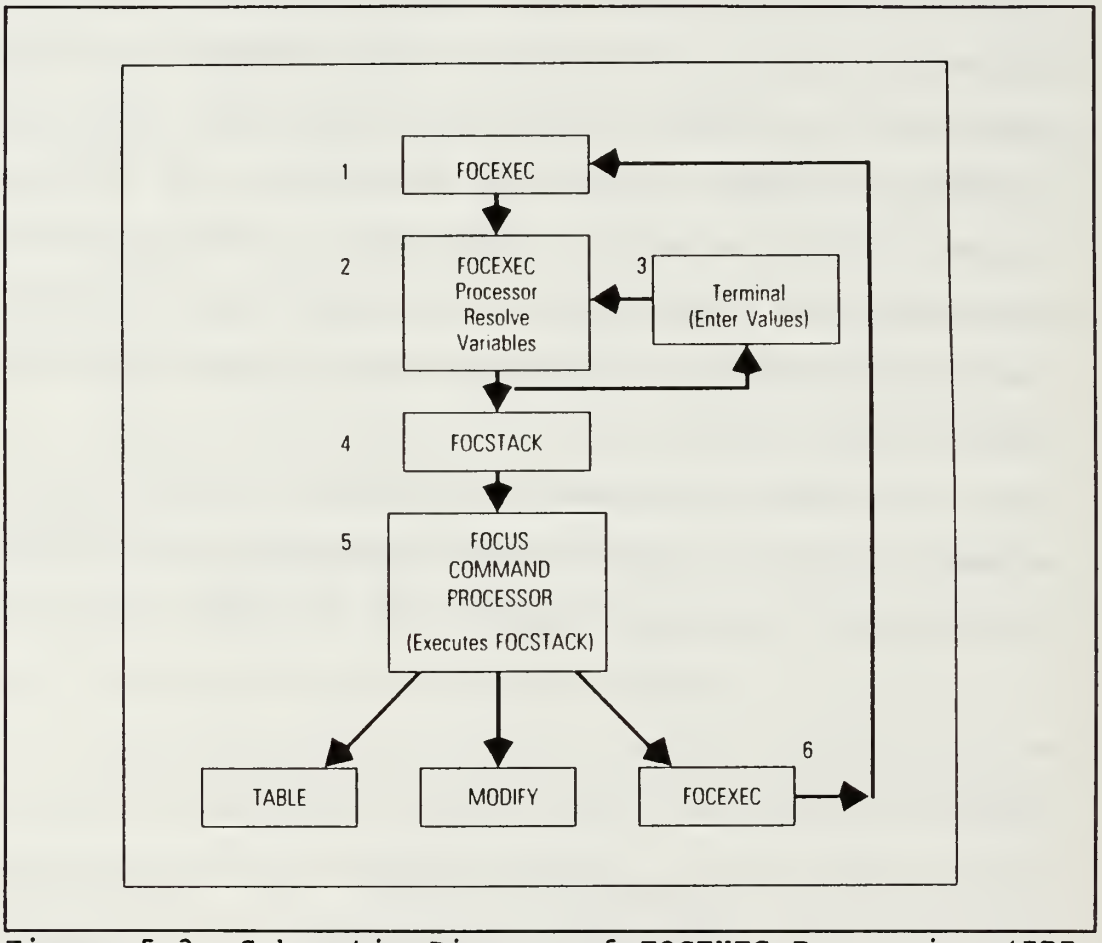


Figure 5.3 Schematic Diagram of FOCEXEC Processing (IBI, User's Manual, Vol. I, 1990, p. 6-10)

The Dialogue Manager also allows interactive variable substitution whereby values for the variables, both global and local, can be supplied during execution or within the FOCEXEC

itself, or from prompts, screens, windows, or menus. The variable can refer to a FOCUS command (such as &option for print or count) or a particular field (such as &city). For system and statistical variables such as the date or number of duplicates, the system automatically supplies them when requested. If the system requires a variable value and it has not been supplied, the system will prompt the user for that value. The values supplied during prompting can be tested and based on the results, procedures can be branched to the appropriate section of the FOCEXEC according to that processing need. The EX or INCLUDE command can also be processed from within a FOCEXEC to incorporate multiple FOCEXEC's.

The Dialogue Manager allows control of the flow and timing of a FOCEXEC's execution through the RUN, EXIT, and QUIT statements. Sections can be processed in any particular sequence and tests can be performed with the results used in subsequent sections or in other FOCEXEC's. Testing values and branching are constructed using arithmetic and logical operators with FOCUS' IF and GOTO commands whereby control can be passed to a user-defined label. Operating control statements, external files, and a library of useful subroutines from FOCUS can also be called. Dialogue Manager and FOCUS commands can be displayed as they are executed in order to test and debug the FOCEXEC's, or if required, only the control logic provided by the Dialogue Manager commands

can be tested by excluding the FOCUS commands. (IBI, User's Manual, Vol. I, 1990)

The evaluation criteria for evaluating dialogue flows is based on how effective the tool conveys and implements flow from one procedure to the next. It is very important that this flow be managed and implemented in such a way that the developer can follow its logic intelligently.

The most obvious difference between IEF and FOCUS is that FOCUS does not separate its dialogue flow from its FOCEXEC's. Dialogue flow in FOCUS is accomplished by using conditional logic like IF and CASE statements and by executing FOCEXEC's within another. As shown in the main FOCEXEC for the MPA system in Appendix C, the Reports Menu can be displayed by selecting option 'R ' from the main menu. In FOCUS, dialogue flow between procedures is illustrated with a menu hierarchy chart or system flow chart as contained in Appendix D.

In IEF, this FOCEXEC would be designed as two procedures: one for the main menu and one for the reports menu, with a corresponding procedure action diagram, screen, and flow defined. Any other procedure such as Minor Property Maintenance that used the Reports procedure would also have a dialogue flow established. These flows are presented as a Dialogue Flow Diagram as shown in Appendix D. This tool is a unique way of showing screen to screen flow but can be difficult to read with many lines and procedures. Perhaps a

procedure structure chart would be more appropriate and more user-friendly.

Dialogue flow as implemented in IEF has its advantages. If a system is packaged as two or more executables, dialogue flow including the passing of data can be transferred from one executable to the other (Penrod, 1992). Commands, function keys, **and** exit states can trigger flow from one procedure to the other.

However, the effort required to define and manage the dialogue flow raises doubt as to its utility for single executable and relatively simple systems. For example, if a developer defines a certain exit state (like FSC not found) to trigger another procedure (like List FSC's), the developer must make certain that each time that exit state is encountered in the source procedure that defined dialogue flow should occur. In FOCUS, if another procedure needed to be executed, an 'ex' command would simply be used. Furthermore, rarely would a procedure be executed by a command, function key, **and** an exit state.

Sometimes processes like List FSC's must be "promoted" to a procedure to establish dialogue flow. This type of procedure is referred to as a designer-added procedure. Once a procedure, it cannot not be used like a process. For example, the Report Procedure could not have a 'Use List FSC' statement; instead a dialogue flow must be defined between the

Report Procedure and the List FSC procedure. Processes, however, can be invoked by many procedures.

FOCUS ignores this distinction as procedures and processes are both implemented as FOCEXEC's. FOCUS also allows the developer to set global and local variables; only local variables can be established in IEF and they are usually implemented as work attributes (Penrod, 1992).

To display the details of the dialogue flow, the developer must laboriously select a series of windows and menus similar to displaying the details of the ERD. A more concise method of displaying the dialogue flow is needed because the documentation is lengthy and difficult to read as illustrated in Appendix D.

The prototyping feature in IEF is really a misnomer. Only the screen flow can be verified; no data can be entered. Prototyping in the RAD context is actually action diagramming and testing in IEF.

Therefore, the extra work required to detail the dialogue flows in IEF seems to complicate rather than simplify procedural flow. The Dialogue Flow diagram while helpful can just as easily be documented as a system flow chart or matrix. Nevertheless, dialogue flow is managed better in IEF. There is no complimentary tool, diagram, or management mechanism in FOCUS. Intelligent dialogue flow and the passing of data must be programmed by the FOCUS developers and then documented.

5. Screen/Report Design

The objectives of IEF's screen design are to standardize the screens across the business system, and to design the screen in such a way that the user finds the screen flow clear and easy to follow. In IEF, screens are only associated with one procedure or procedure step and are initially named after the procedure or procedure step they implement. The Screen Design Tool is used to create templates as well as screens. Templates are usually sections of a screen that can be used by many other screens across a business system in order to establish standardization and consistency for the user interface. Screens are the user's view of the system and provide **both** data **to** the procedure's import view and data **from** the export view. Screens can also be accessed from the dialogue flow diagram or action diagram by chaining.

Screens can be created automatically or step by step. The layout feature in screen design automatically designs a screen based on the import and export views and initial screen defaults defined for the business system. The layout feature also gives the developer the option of using a template(s).

Templates and screens may contain fields, which are the attributes of the import and export data views; literals which is simply text; special fields (which can be supplied by IEF that are not derived from the business system but are introduced for a specific purpose like the date and time); and

prompts which are the labels for the fields and special fields. Screens can be customized by specifying the location and characteristics of the attributes such as their import or export data view, their display length, the number of decimal points for numeric attributes, whether the attributes are hidden or not, their edit patterns such as XXX-XX-XXXX for a social security number, their video display properties (color, protected, intensity, highlight, justification, fill characters, etc.), their prompts if applicable, the error video display properties (to highlight a field in error if not a permitted value), and their help identifiers. This last characteristic is used to integrate an attribute with a database management system's help facility which would link that identifier with a help description; IEF provides the "hooks" but not the facility (Penrod, 1992). Prompt names are assigned to fields so they are consistent across screens. Special fields include a PF key line which reserves an area of the screen in order to display the function keys and their associative commands, a system error message line, a command area for the user to input the appropriate command, the current date and time, and scroll bar messages such line xx of xx lines, among others. Appendix E presents the screen for Minor Property Maintenance.

Repeating and nested repeating groups or lists can also be displayed. Repeating group properties can be defined to allow scrolling, to prevent the updating or the addition of

data to the screen, and to determine if an occurrence (or record) on the list should be displayed upon returning from the link, or if the original screen should be re-displayed. Appendix E contains a screen with a repeating group view and a selection work attribute which is used to specify the action, such as select, to be performed on the occurrence. Commands can be assigned to function keys for a specific screen so that the appropriate action block is executed. For example, F6 will initiate the List FSC action block so the user can obtain and select the appropriate FSC from the list.

Moving, copying, centering, and deleting fields is easily accomplished by selecting the option and then "rubberbanding" the field. When deleting fields from the template, the deletion applies across all screens that use that template. The Screen Design Tool also has a display option which displays the screen as it appears to the user; a consistency check for the screen; and a 'Used In' option to determine which screens are using which template(s). (TI, Rapid Development Using the IEF, 1991)

Screen design is performed in FOCUS by the FOCUS Interactive Data Entry Language (FIDEL) and can be used with both MODIFY and the Dialogue Manager. Alternatively, the developer can use the FOCUS Screen Painter to interactively build and view the screen online and automatically create the FIDEL code. A screen is created by using the CRTFORM or -CRTFORM command of Dialogue Manager which both invoke FIDEL.

FIDEL is frequently used with the MODIFY facility for setting up screens for database maintenance and with the Dialogue Manager primarily when variables need to be entered.

The developer can specify three types of fields on the screen: input, display only, and "turnaround" (both display and update). Display values are considered protected. PF key controls and cursor positioning can be set a well as the screen attributes such as highlighting and color. Labels, attributes, and field formats are part of the definition of a particular field. For example, -"<T.HIGH.&CITY/7" refers to a highlighted field variable called city with a length of 7 that can be displayed and updated.

Multiple forms can be displayed on the screen as can multiple occurrences in order to update many values at once rather than one at a time. The screen will handle several errors such as a format errors, for example, when entering non-numeric data for a numeric field; validation errors when the input values fails the validate test coded in MODIFY; NOMATCH errors when the data entered did not match a record in the file; DUPLICATE errors when the record already exists; and ACCEPT errors when the input value failed the ACCEPT test. To capture all of the data on the screen, the input values can even be logged to a file.

From within a FOCEXEC that contains a CRTFORM, the PAINT command will execute the Screen Painter. The FOCUS Screen Painter can be used to design a full-screen layout by

placing literal text and areas for fields on the screen in any position. The field areas are then assigned to database or computed fields by typing in the field name and selecting the screen attributes. Like IEF, the screen can also be viewed as it appears to the user from within the Screen Painter or within TED. FOCUS will automatically code the CRTFORM and can generate a CRTFORM that contains **all** the fields in the master file description, similar to the layout function in IEF. (IBI, User's Manual, Vol. I, 1990)

A screen or crtform that requires specific data input for a procedure is usually incorporated into the FOCEXEC that contains the logic, unlike IEF, which has a separate screen design facility to manage the screens. The main menus in FOCUS, however, may be consolidated into one FOCEXEC. Appendix C presents the MPA Maintenance Menu screen in which the user selects a menu option, and the appropriate FOCEXEC, located as a separate file, is executed.

IEF does not have a report writer. An export view with a repeating group can be used to display a list of records and then printed or saved to a file for later printing. FOCUS, on the other hand, has a report facility called TABLE. A report request consists of the key word TABLE followed by the name of the file; a command verb such as print, list, count or sum; the field names; any calculations with the COMPUTE and DEFINE expressions; and any modifiers that control the selection, sorting and formatting of the

data. When entering TABLE requests, online error correction and help are automatically performed.

FOCUS also has a facility called Table Talk, primarily designed for the end user, in which table requests can be built by selecting from separate windows the file to be used, the sort criteria, etc. It "walks" the user through the request similar to machine-led dialogue in IEF. Any adhoc requests not written into the system can easily be executed by the user, instead of the developer, with Table Talk.

Using TABLE, data can be displayed by groups, sub-totaled, ranked, counted, or tested before or after execution, as well as displayed in many other display formats that are too exhaustive to list. Instead, as an example, total sales within a city, within a state, and compared to total overall sales can be executed with one sentence. The edit function can be used to convert fields or to extract characters from an alphanumeric string, or to insert characters to form a new variable. FOCUS will automatically format reports but these defaults can be overridden in order to customize a report. The user can also select whether records with missing values and their children should be printed or not.

A table request can include fields from other segments in the file but they must be stated in a top-bottom path. FOCUS also provides the JOIN and MATCH commands to join multiple files for reporting purposes. Even the retrieval process can be changed by requesting an "alternative view" of

the database by specifying which field name should be the root. An especially wide report can be divided into panels so that each panel can fit on the screen with scrolling in all directions. Finally FOCUS provides a library of subroutines which can be "called" on to execute any number of functions and utilities, especially numeric functions.

To evaluate screen and report design requires determining how effectively the tool builds and presents screens to the user and how the tool integrates the screens and reports with the programs. IEF has its own separate screen management facility and allows the use of templates. In FOCUS, the developers simply copy a standard CRTFORM as their template and modify it accordingly. The screens in FOCUS, except for the main menus, are incorporated into the FOCEXEC with the programming logic. They are usually created in the beginning of the program so the user can enter key fields to retrieve the correct record, and then another screen or report is displayed for additional data input or for informational purposes respectively. The FOCEXEC for updating minor property contained in Appendix C is an example of incorporating forms into a FOCEXEC.

IEF implements one screen for all functions (create, display, delete and update) and uses the same screen for importing and exporting data. As such, IEF assumes the user knows what key fields to enter for each type of action. Screens cannot be associated with processes, only with

procedures. Therefore, if a process such as adding a minor property item required its own screen, for whatever reason, the process has to be "promoted" to a procedure and the corresponding dialogue flow defined. Unfortunately, some of the processes may use the procedure screen and others may not. For example, each report or list is usually defined as a procedure since the export screen is not the same as the import screen. This requirement, to create a procedure from a process in order to customize a screen for a process, creates more work than necessary especially when the dialogue flow must also be defined. IEF could create a similar screen management facility for screens of **processes** similar to its screen facility for procedures. FOCUS allows the creation of screens whenever and wherever they are needed.

The separation of the screens from the procedure action diagrams does allow management of the screens but compared to process and procedure management, screen management is certainly not as important to the developer. In IEF the screens are logically connected to the procedure action diagrams through views: all import, export and local views of the procedure are automatically mapped to the procedure screen. For this reason, no extraneous attributes are presented on the screen. However, when performing screen maintenance such as adding an attribute, the developer must first return to the PAD and add the attribute to the appropriate view before selecting it during screen design.

In FOCUS, as long as the correct file is accessed, all fields from all segments can be added and deleted from a screen. No checking as to whether certain fields should be included or not is performed; it is assumed that the FOCUS developer knows the logic of the FOCEXEC when designing the screen. It appears IEF includes this logical check with the PAD to prevent bad screen design (Penrod, 1992). The author gives the developers more credit. In practice, FOCUS developers usually use TED to create their CRTFORM's and then use the PAINT command to view how the screen would appear to the user (Harr, 1992).

Both tools can automatically layout a screen although IEF's screen painter is superior. It is more user-friendly, can manipulate fields, literals, etc. with ease, and the developer can build and customize screens quicker than in FOCUS.

One of the major differences between IEF and FOCUS appears in the reporting facility. IEF does not include an inherent reporting facility, except for repeating groups on screens; instead it relies on the implemented database management system or a third-party product for its reporting function. This reasoning is based, in part, on the fact that IEF is usually installed in large organizations that already have extensive reporting techniques. Since the logical model is separated from the actual data, end user interaction, similar to creating adhoc reports as implemented in FOCUS with

Table Talk, is not available in IEF. With respect to offering a totally integrated and comprehensive product and with respect to implementing IEF for a new or small organization, the lack of a reporting capability is a major deficiency. The same can be stated for IEF's lack of numeric functions or a library of commonly-used action blocks. There is no count, sum, subtotal, or total commands! The IEF developer must code the calculations as action blocks. Figures 5.4 and 5.5 present the difference between IEF and FOCUS for a simple count operation. Finally, the screen displays in IEF are limited to the width of the screen whereas in FOCUS, panelling from right to left will allow extra wide screen displays.

```
TABLE FILE MINOR
CNT.TAG_NO IN 40
```

Figure 5.4 Count Function in FOCUS

6. Documentation, Training, and Technical Support

IEF's documentation consists of an analysis and design toolset guide. These guides provide a short introduction of each tool and then step-by-step instructions on how to access the tools and perform the desired function. It does not provide the underlying concepts or integration features as presented in the IEF development tutorial. Even the examples in the "Rapid Development Using the IEF" guide did not

```
CALCULATE_NUMBER_OF_PROPERTY
IMPORTS:
    Entity view import custodian (Mandatory)
        cust_code (Mandatory)
EXPORTS:
    Entity view export custodian
    number_of_property
LOCALS:
ENTITY ACTIONS:
    Entity view property
    number

READ EACH property
    WHERE DESIRED property is signed for by
        import custodian
SET export custodian number_of_property
    TO export
    custodian number_of_property + 1
```

Figure 5.5 Count Function in IEF

correspond with the tutorial exercise.[11] However, the tutorial exercise was most instructive and an excellent training tool. What IEF needs to publish is an application developer guide to IEF that includes the concepts, tips, and techniques of constructing effective IEF models. FOCUS' users manuals, Volumes I and II, are quite user-friendly and helpful. The manuals thoroughly explain the commands with ample diagrams and examples, many of which are explained line by line.

IEF assumes that at least one person from the organization attend its training courses for all the toolsets purchased. To learn the entire tool, IEF's FasTrack program

---

[11] The guide and tutorial exercise is still in beta development as of February 1991.

which requires a minimum training period of 21 days at a rate of $500 a day ($10,500 total) is recommended (TI, Education Schedule, 1992). Whereas large organizations may be able to afford this cost, small organizations may not. To really understand the tool, the training courses are essential. Many sites obtain information engineering methodology training in addition to IEF training, and implement just-in-time training so they learn the appropriate tool prior to developing their own system. (McGrail, 1992) Often, TI on-site consultants are contracted for initial assistance. Otherwise, the users are referred to a product specialist or their account representative for help. IEF's hotline support is to be utilized for technical problems and fatal errors. FOCUS' hotline support, on the other hand, fields basic questions but will not, understandably, program the user's FOCEXEC's.

D. METHODOLOGY AND TOOL SUPPORT FOR THE METHODOLOGY EVALUATION

To evaluate the two methodologies requires not only analyzing the methodologies themselves, but also analyzing how the tool supports the methodology and the tool's "fit" with the software environment. The advantages claimed of CASE tools such as methodology integration, enforcement, standardization, and other qualities are analyzed to determine if the claims are true for IEF. The costs and benefits are then analyzed against the current methodology and software

development environment to determine whether the introduction of a CASE tool is worth it.

James Martin states that employing RAD implies using a CASE tool. But is a CASE tool really necessary? Can a fourth generation language accomplish RAD better? Or is information engineering a more suitable methodology?

Information engineering (IE) is defined as "the application of an integrated set of formal techniques for the planning, analysis, design, and construction of information systems" (Martin, 1991, p. 1). With respect to the analysis and design phases, IE and rapid application development (RAD) as employed by MIS are very similar. As mentioned before, James Martin included RAD as an alternative pathway through the IE cycle. Both methodologies perform data modelling, systems analysis, and design. Each constructs executable systems by using a high-level programming language with integrated control, processing logic, and screen/report design. True, RAD usually employs smaller teams, encourages active user involvement, has shorter completion times, and emphasizes quick prototyping. But the concepts of IE and RAD, without any reference to the tools that support the methodologies, are practically the same. The differences lie in the implementation, integration, and enforcement of the methodologies.

IE and IEF go hand-in-hand. IEF was designed to enforce a single methodology. This strategy is immediately apparent

because the stages of IE are listed as menu choices on the second screen of IEF. There are tools that support each of the phases, and these tools can be accessed from one phase or another or from one tool to another through chaining. For example, from the PAD screen the developer can chain to screen design or dialogue flow. Information entered with one tool is logically linked to the other tools: the information in the data model is used to create the views and the processing logic in the PAD, the views are incorporated into the screens, the processes can be used to create the PAD's, and the reports are in sync with the model. More importantly, all the information contained in each of the phases is incorporated into **one** model.

This integration enforces a sequence to application development with IEF. The data model must exist before creating process action diagrams; views must exist before logic can be added to the action diagrams; views must exist before screens with attributes can be created; a business system must be defined before procedures can be created; and procedures must exist before detailing the dialogue flow. IEF even includes, in the toolset documentation, a build action diagram and create views activity hierarchy (flow chart) to assist the developer. This integration implies a data model to process action diagramming to business system definition to procedure creation to screen design to dialogue flow sequence, at least initially. Except for situations where this sequence

is enforced, the developer still has the flexibility to chain from one tool to the other ignoring the recommended sequence.

Enforcement also occurs when adding or deleting items. For example, an attribute cannot be deleted from the ERD until that attribute is deleted from all PAD's; a view cannot be deleting from a PAD if the processing logic refers to that view; and a process cannot be deleted until it is deleted from all procedures that use that process.

Consistency checks also enforce the methodology by providing the "green light" before preceding to the next stage. The model must be consistent before transformation and before generating code. Otherwise, the developer can still work from tool to tool or from phase to phase even if errors occur. Consistency checks can be performed at the toolset, phase, or model level so the developer can incrementally check his/her work. These errors are further classified as fatal errors which usually require IEF technical support; errors which indicate a condition that must be corrected before code generation; severe warnings which indicate conditions that will cause errors during system implementation; and warnings which indicate conditions that should be corrected but will not interfere with future work. IEF also enforces standardization through common menu screens and options (detail, check, chain), through common diagramming manipulation techniques (locator, expand, contract), and through business system defaults and screen templates.

The RAD methodology as employed by MIS consists of systems analysis and data-driven prototyping. The methodology is enforced by the completion of required feasibility and system analysis documentation and by the construction of an evolving prototype. With the exception of a review of the master file description, the developers are left to their own skills and techniques to design the system. Some standards are enforced: screens are built with the same format within a system and often across systems; FOCEXEC's are constructed following structured programming techniques; and FOCEXEC's and reports are named according to a functional schema (Nolan, 1992). A user's guide and database manager's guide is required but no standard format is followed (Harr, 1992).

FOCUS supports the RAD methodology by providing the data modeling, design (screen and report), programming, and database management facilities. Specifically, these are the MFD, TED, Dialogue Manager, MODIFY, FIDEL and Screen Painter, TABLE, and FSCAN facilities. FOCUS never claimed to be a comprehensive integrated CASE tool; it is simply a fourth generation programming language. Nevertheless, integration is achieved with the tools because screen design, dialogue flow, database management, and report requests can all be incorporated in one FOCEXEC. Integration is not supported between analysis and design or with the documentation of the system. Prototyping in RAD is easily accomplished with FOCUS: sample screens and reports can be developed very quickly and

sample screens and reports can be developed very quickly and then iteratively refined.

An integrated CASE tool that supports a single methodology from planning to implementation has its advantages. Training of the tool and methodology go hand-in-hand with the developer understanding how the methodology is applied with the tool. The terminology is the same. Certainly having all the information stored in **one** model is an advantage.

With respect to analysis and design, the only phase not implemented in FOCUS is activity analysis. Yet even within IEF, activity analysis is not enforced and is not required for code generation. RAD as employed by MIS requires the same systems analysis but does not employ tools to assist in the analysis. If the system is complex, having an analysis tool may be considered a benefit, if the results outweigh the time invested. The integration factor for activity analysis with the rest of the model is low -- only process transformation, which can be used to initially create the process action diagrams, is integrated with activity analysis, and stereotyping without activity analysis achieves the same results.

What are the advantages and disadvantages to IEF's enforcement of the IE methodology? Both IEF and FOCUS require a data model and MFD respectively before programming. Whereas IEF may **stop** the developer in situations where the methodology is enforced (before database transformation and code

generation), FOCUS will respond with an error message **after** execution of the FOCEXEC (except for TABLE requests which have on-line error correction). Is this enforced integration a help or a hindrance? The answer depends on the "worth" or reason for the integration in the first place and the tradeoff in preventing an error versus catching it later on. For example, the machine-led dialogue in which the developer has to click on each word or phrase while performing action diagramming may prevent some syntax and logic errors but will not prevent bad programming. The consistency check, rather than machine-led dialogue, should suffice in catching errors. Having to specify the views in the procedure action diagram before screen design hinders design as does the requirement to consolidate all views in the procedure action diagram. Experimentation and "what-if" scenarios are very difficult to implement with these restrictions. On the other hand, the enforcement of deleting items until all references to that item are deleted, does ensure a "clean" model.

IEF does **not** enforce a certain sequence of application development tasks for many of its tools. This flexibility acknowledges the fact that application developers do not work in sequence. They jump from tool to tool as new logic, processes, or requirements occur or change. To paraphrase Page-Jones, 1992, p. 36:

> Developers often shift opportunistically and unpredictably among different components of the system and at different levels of detail. These components or subsystems can be

> in different states at a particular time and the developer may return to the same point with a different perspective or idea. The tool should expect the human to build up system understanding like a jigsaw. In general, current CASE tools do not support this degree of freedom in a robust and consistent manner.

IEF and FOCUS both support this flexibility but it comes with a price: more errors. The developer is just as susceptible to creating errors in IEF as in FOCUS as the author discovered. IEF does not stop application development after a certain number of errors although it is recommended to perform consistency checks or with FOCUS, to execute the FOCEXEC before preceding to the next step. True, IEF's consistency checks on separate elements of the model such as a screen or dialogue flow diagram assist in isolating errors; with FOCUS, it is difficult to determine where the error exists if the FOCEXEC "calls" other FOCEXEC's (Harr, 1992). Therefore, there is no proof that IEF produces better quality systems than FOCUS through enforcement of the IE methodology.

Others claim that IEF adds structure to application development. Perhaps this viewpoint reflects the fact that IEF is organized: it divides application development into separate tasks with a management facility to organize it. The developer can list all screens, action diagrams, procedures, and processes, for example. With FOCUS, everything but the data model is usually integrated into the FOCEXEC. It is the responsibility of the programmer to apply sound, structured

programming techniques to prevent haphazardly written programs.

Whether this separation in IEF is an advantage or not depends on the price paid for the separation and the benefit gained. For dialogue flow the price is too high, but for screen design it is not. Perhaps the organizational utilities of IEF should be added to FOCUS.

Automatic diagram and report generation would seem a benefit for IEF since any changes to the model are instantly reflected. Yet, IEF should not be judged better simply because it has advanced and integrated diagramming and report generation facilities; rather it should be judged on the usefulness of the diagrams and the reports.

The time invested in creating some of the diagrams (in particular the activity dependency diagram) should be evaluated against their utility especially if the diagrams are only going to be used for documentation. IEF, as an integrated CASE product, is more than a tool to draw pictures. The reports, many of which are lengthy and not user-friendly, should not be generated if they are not going to be used. In comparison to FOCUS, IEF's structure chart, screen design facility, and action block usage chart are features that would prove useful to any developer and represent IEF's strength in organizing program structure. All other reports and diagrams have not proven their utility compared to alternative methods used by the FOCUS developers. Note, also, that most of the

diagrams and reports are for the system developer; many of the changes, with the possible exception of the structure chart, are **not** automatically reflected in the **user's** guide. Neither tool can substitute for accurate and well-written user instructions and descriptions.

IEF also claims to enhance user communication especially through the use of its diagrams. Unless the users are trained in the concepts and terminology of data modelling and/or design, whether in IEF or FOCUS, no effective user communication can occur. The diagramming manipulation techniques of IEF, especially for the data model and screen design, do allow changes to be reflected on the screen quickly for the user.

Nevertheless, the best technique to enhance and refine the requirements of a relatively simple system is to actually demonstrate the system, to interactively prototype. "Analysis paralysis" or spending too much time with the model should be avoided since the best analysis and design techniques will still not uncover all the changes discovered during prototyping. The secret to achieving complete and accurate system analysis and design lies with the questions asked by the developer and the developer's flexibility to cope with constant change.

Prototyping in IEF is restricted to screen prototyping since data entry cannot be performed prior to coding. With FOCUS, the developer "runs' the program ignoring errors that

are not relevant to the testing being performed. Consider the difference in the work required to build an initial prototype between the two tools. In IEF, the minimum work required for constructing a system is a data model, process action diagram, a procedure action diagram, a dialogue flow diagram, a screen design, and load module packaging (how many executables are to be created) as presented in Appendix F. And then, **all** errors must be corrected before coding and testing. With FOCUS, one "menu" FOCEXEC with control logic and a screen, and at least one "called" FOCEXEC with its screen(s) and program is required. No compiling or loading of code is necessary. Error-free code is not a prerequisite to prototyping in FOCUS. FOCUS therefore can prototype a system faster than IEF even if stereotyping is used.

There may be some truth to the claim that with IEF more attention is spent on analysis and design than on coding. This is definitely true if a lot of time is spent on activity analysis. Moreover, the model is more or less conceptual until the construction phase when code is actually generated. Traditional coding, although at a higher level, does occur when creating and modifying the action diagrams. In reality some of the coding like screen design has been replaced by diagramming tools in IEF. This emphasis on analysis and design is a step in the right direction.

A computerized methodology also offers advantages: the methodology can be integrated with the tools as it is with

IEF; the methodology may have built-in computer-based training; project management tools may be built into the computerized methodology; the methodology is standardized; and the methodology may be enforced through automation.

E. CONCLUSION

Both the analysis and design phases of information engineering (IE) and rapid application development (RAD) and their respective tools, IEF and FOCUS, have been evaluated with respect to the software development environment of the MIS department. The results are based on literature reviews, interviews, and on a case study of the Minor Property Accountability System which was developed with each tool and methodology. The answer to the primary research question posed by this thesis, the costs and benefits of introducing information engineering and IEF compared to the current rapid application development methodology and fourth generation language FOCUS for analysis and design, are presented in the following sections and in Figure 5.6.

Both methodologies perform data modelling and systems analysis. Both construct executable systems by using a high-level programming language with integrated control, processing logic, and screen/report design. RAD usually employs smaller teams, encourages active user involvement, has shorter completion times, and emphasizes rapid prototyping. But the concepts of IE and RAD, without any reference to the tools

that support the methodologies, are practically the same for analysis and design. The differences lie in the implementation, integration, and enforcement of the methodologies through their tools.

The benefits of IEF range from its support of its methodology to specific tools. First, IEF implements a single methodology, IE, which implies a sequence of application development. For some organizations, linking the tool with the methodology adds structure to the development process. IEF enforces the methodology by requiring a consistent model before database transformation and before generating code. This same enforcement, however, hinders quick prototyping. Machine-led dialogue and consistency checks are used to prevent errors. Enforcement **within** the tools is **not** strictly enforced, allowing the developer the flexibility to move from tool to tool or from phase to phase as needed. Integration is achieved by incorporated all the information into **one** model. Most changes are automatically reflected throughout the model including the model reports. More emphasis is placed on analysis and design than coding because IEF implements some of its tools through diagramming facilities and the developer never touches the code; it is instead generated. A computerized methodology offers potential integration with automated project management software, standardization, and computer-based training. Standardization within IEF is

achieved through consistent options and menus, through screen templates, and through business system defaults.

As for IEF's tools, the screen design facility; organizational tools for managing screens, processes and procedures; and the structure chart and action block usage diagrams are its strengths. Its activity analysis tools are effective only for complex systems. The work required for view maintenance and dialogue flow are considered costs with respect to rapid application development with IEF.

IEF is really not, in this author's opinion, a complete and comprehensive CASE tool because it separates the model from its targeted database facility and relies on the database facility to perform some of its functions. Whereas these functions may be separated in a large organization, for the MIS department, they are not. The application developer is the analyst, programmer, and database manager. For this reason, IEF does not have an inherent security facility, a report generator, or an action diagramming language with numeric functions. The absence of these features is a major deficiency when compared to FOCUS.

FOCUS supports the RAD methodology through its quick and easy execution of data-driven prototyping. The current methodology works well for the MIS department given the level of experienced application developers, the small size of the organization, the relative limited complexity of the systems developed, and the fact that there is no need to trace

requirements to code. The absence of an enforced application development standard is a weakness of MIS' RAD methodology.

FOCUS is a higher level programming language (it can do more with less code) than the action diagramming language of IEF. FOCUS can incorporate screen design, programming logic, and dialogue flow in one executable procedure or FOCEXEC. Its database management facilities are also integrated: security can be incorporated within the data model and the database can be scanned, graphed, and invoked for user adhoc reports. FOCUS' screen design tool and lack of management facilities to track program flow are its weaknesses.

Therefore, IEF offers the following benefits compared to FOCUS: a one model implementation, a standard computerized methodology, consistency checking to a low level of detail, management tools for the developer, standardization throughout the model, and superior diagramming features and screen design. FOCUS offers the following benefits compared to IEF: rapid prototyping, a higher level programming language, a report facility, security within the data model, inherent database management facilities, and excellent documentation. Neither tool is better than the other in terms of integration, enforcement and support of the methodology, system or model documentation, enhanced user communication, activity analysis, dialogue flow, or training/hotline support.

The benefits of IEF do not outweigh its costs. With respect to the application development environment of the MIS

CRITERIA	IEF	FOCUS
ONE MODEL CONCEPT	X	
COMPUTERIZED METHODOLOGY	X	
PROTOTYPING		X
INTEGRATION	---	---
CONSISTENCY CHECKING	X	
METHODLOLOGY ENFORCEMENT	---	---
STANDARDIZATION	X	
ENHANCED USER COMMUNICATION	---	---
DATA MODELLING		X
ACTIVITY ANALYSIS	---	---
PROGRAMMING		X
DIALOG FLOW	---	---
ORGANIZATIONAL TOOLS	X	
SCREEN DESIGN	X	
SYSTEM DOCUMENTATION	---	---
REPORT FACILITY		X
SECURITY		X
TRAINING/HOTLINE SUPPORT	---	---
DOCUMENTATION		X
DATABASE UTILITIES		X
CULTURAL IMPACT		X

Figure 5.6 Comparison of IEF and FOCUS

department, the requirement to rapidly prototype (especially to program and generate reports), to incorporate security easily within the data model, and to interface directly with

the database clearly supports RAD and FOCUS as the application development methodology and tool of choice for analysis and design for the MIS department.

Further areas of research include extending the evaluation to the information strategy planning (ISP) and technical design (TD) and construction phases. In particular, the costs and benefits of extensive system maintenance and testing could be evaluated as well as the implementation and cultural impact of introducing IEF as a shared, distributed model.

APPENDIX A: DATA MODELLING

Model : MINOR PROPERTY ACCOUNTABILITY Sept. 08, 1992 13:16
Subset: (complete model) page 1

Entity Definition

Entity: PROPERTY

Description: Minor property information that uniquely identifies a piece of equipment from $300 to $5000 exclusive and associates it with the purchasing information

Subject area: MINOR_PROPERTY_ACCOUNTABILITY

Properties: Min Occ: 15000 Avg Occ: 35000
Max Occ: 75000 Growth Rate: 20% per year

Attributes: TAG_NUMBER
ORIGINAL_COST
CURRENT_COST
SN
MODEL_VERSION
YEAR_ACQUIRED
INVENTORY_DATE
BLDG
ROOM
MFR
SOURCE_DOCUMENT
PO
PROPERTY_PASS_NUMBER
REMARKS
ACTION

Relationships:
Always SIGNED_FOR one CUSTODIAN
can transfer.
Always IS_ASSIGNED one FSC
cannot transfer.

Identifiers:
1 TAG_NUMBER

Entity Definition Report for the Property Entity (IEF)

```
FILE=MINOR,SUFFIX=FOC
SEGNAME=DEPT,SEGTYPE=S1
  FIELDNAME=DEPT_CODE     ,ALIAS=DEPTCD ,FORMAT=A5    ,DESC='DEPARTMENT CODE',$
SEGNAME=CUSTN,PARENT=DEPT,SEGTYPE=S1
FIELDNAME=C_MAILCODE,ALIAS=CUSTNMC,FORMAT=A4     ,DESC='CUSTN MAIL CODE'     ,$
FIELDNAME=C_LST_NM  ,ALIAS=CUSTNLN,FORMAT=A20    ,DESC='CUSTN LAST NAME'     ,$
FIELDNAME=C_FRST_NM ,ALIAS=CUSTNFN,FORMAT=A10    ,DESC='CUSTN FIRST NAME'    ,$
FIELDNAME=C_PHONE_NO,ALIAS=CPHONE ,FORMAT=A7     ,DESC='CUSTN PHONE NO'      ,$
  DEFINE DEF_PHONE/A8=EDIT(C_PHONE_NO,'999-9999')                              ;$
SEGNAME=MINRPROP,PARENT=CUSTN,SEGTYPE=S1
 FIELDNAME=TAG_NO         ,ALIAS=TAGNO   ,FORMAT=A7     ,DESC='TAG NUMBER'
    FIELDTYPE=I ,$
 FIELDNAME=PUR_REQ_NO     ,ALIAS=PURREQ ,FORMAT=A7     ,DESC='PURCHASE REQ'
    FIELDTYPE=I ,$
 FIELDNAME=SOURCE_DOCNO   ,ALIAS=DOCNO   ,FORMAT=A10    ,DESC='PURCHASE ORDER' ,$
 FIELDNAME=FSC_CODE       ,ALIAS=FSCNO   ,FORMAT=A4     ,DESC='FED STOCK CODE'
    FIELDTYPE=I ,$
 FIELDNAME=ACQ_DATE       ,ALIAS=ACQDT   ,FORMAT=Y      ,DESC='Y-YEAR ACQUIRED',$
 FIELDNAME=ACQ_COST       ,ALIAS=ACQCOST,FORMAT=D6.2   ,DESC='INITIAL COST'    ,$
 FIELDNAME=MANF_MOD_NO    ,ALIAS=MODNO   ,FORMAT=A12    ,DESC='MANUFACTURE MOD#',$
 FIELDNAME=MANF_SER_NO    ,ALIAS=SERNO   ,FORMAT=A16    ,DESC='MANUFACTURE SER#'
     FIELDTYPE=I ,$
 FIELDNAME=MANF_NAME      ,ALIAS=MFGNM   ,FORMAT=A25    ,DESC='MANUFACTURE '    ,$
 FIELDNAME=ITEM_DESC      ,ALIAS=IDESC   ,FORMAT=A35    ,DESC='NOMENCLATURE'    ,$
 FIELDNAME=ITEM_BLDG_NO   ,ALIAS=IBLDG   ,FORMAT=A4     ,DESC='ITEM BLDG LOC'   ,$
 FIELDNAME=ITEM_ROOM_NO   ,ALIAS=IROOM   ,FORMAT=A4     ,DESC='ITEM ROOM LOC'   ,$
 FIELDNAME=INV_DATE       ,ALIAS=INVDT   ,FORMAT=YMD    ,DESC='INVENTORY YMD'   ,$
     HELPMESSAGE = 'THE FORMAT FOR INVENTORY DATE IS YMD'                      ,$
 FIELDNAME=INV_ADJ_COST   ,ALIAS=ADJCOST,FORMAT=D6.2   ,DESC='ADJUSTED COST'   ,$
 FIELDNAME=INV_RMKS       ,ALIAS=INVRMKS,FORMAT=A22    ,DESC='REMARKS RE INV' ,$
 FIELDNAME=EX_CODE        ,ALIAS=EXCD    ,FORMAT=A1     ,
    ACCEPT = A B C F H T,DESC='EXCESS PROPERTY'                               ,$
 FIELDNAME=ADP_CODE       ,ALIAS=ADPCD   ,FORMAT=A1     ,
    ACCEPT = 4 5 6 7 8 9 X,DESC='CONDITION CODE'                              ,$
 FIELDNAME=AIS_CODE      ,ALIAS=AISCD   ,FORMAT=A10    ,DESC='AUTOMATED SYSTEM',$
 FIELDNAME=SUB_CODE       ,ALIAS=SUBCD   ,FORMAT=A10    ,DESC='SUBSYSTEM CODE' ,$
 FIELDNAME=LAB_CODE      ,ALIAS=LABCD   ,FORMAT=A2     ,DESC='USED BY EC DEPT ',$
 FIELDNAME=MANF_YR        ,ALIAS=MFGYR   ,FORMAT=Y      ,DESC='YEAR OF MANF'    ,$
SEGNAME=FSCSEG,SEGTYPE=KU,PARENT=MINRPROP,CRFILE=FSCFILE,CRKEY=FSC_CODE    END

DBA=XXXXXXXX,$
USER=XXXXXXXX,ACCESS=RW,$
USER=XXXXXXXX,ACCESS=R,$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ 'EC' OR 'EL',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
              VALUE=DEPT_CODE EQ 'PH',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ 'OC',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ 'MR',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ '03' OR '034' OR '034H' OR '035' OR '036',$
           VALUE=OR '0362' OR '0363' OR '0364' OR '037',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ '05' OR '53' OR '54',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ '74',$
USER=XXXXXXXX,ACCESS=RW,RESTRICT=VALUE,NAME=DEPT,
             VALUE=DEPT_CODE EQ '81',$
```

Attribute Definition

Attribute: TAG_NUMBER

Subject Area: MINOR_PROPERTY_ACCOUNTABILITY
Entity Type: PROPERTY

Description: Minor property tag number sticker affixed to the equipment

Properties: Mandatory Basic Text
Length: 12

Default: none

Attribute Definition

Attribute: ORIGINAL_COST

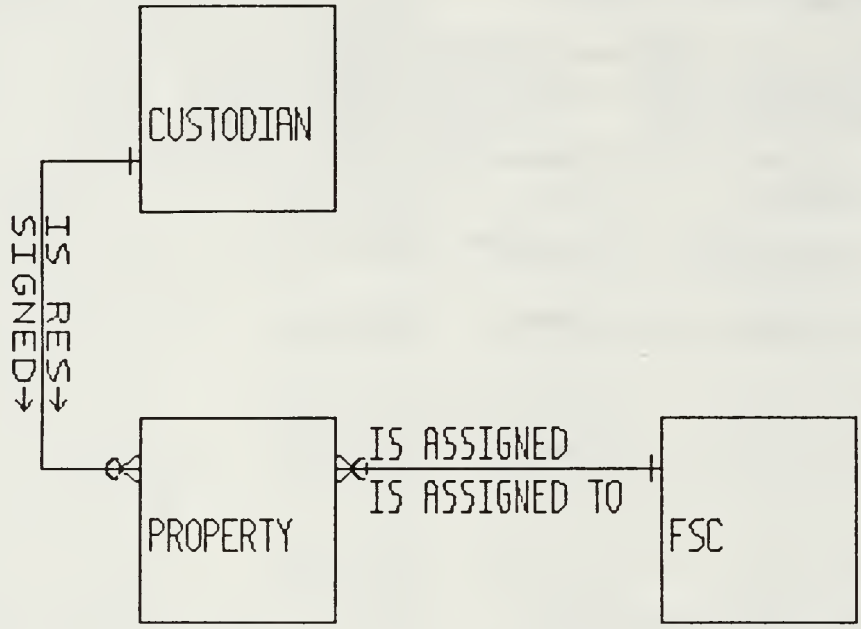
Subject Area: MINOR_PROPERTY_ACCOUNTABILITY
Entity Type: PROPERTY

Description: Original cost as it appears on the purchase order; does not include any additions or deletions to the system; can be unknown if not the original owner

Properties: Mandatory Basic Number
Length: 6 Decimal places: 0

Default: none

Attribute Defintion Report for Tag_Number and Original_Cost (IEF)



Model name:	MINOR PROPERTY ACCOUNTABILITY	Sept. 10, 1992	ERD
Subset name:	ALL	11:06	Page: 1

Entity Relationship Diagram for the MPA System (IEF)

```
 CHECK FILE MINOR PICTURE

0 NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   4  ( REAL=    3  VIRTUAL=   1 )
  NUMBER OF FIELDS=    29  INDEXES=   4  FILES=     2
  NUMBER OF DEFINES=    1
  TOTAL LENGTH OF ALL FIELDS=  315
1SECTION 01

             STRUCTURE OF FOCUS    FILE MINOR    ON 08/25/92 AT 13.36.17


                        DEPT
                     01      S1
                    ************
                   *DEPT_CODE   **
                   *            **
                   *            **
                   *            **
                   *            **
                   **************
                    **************
                          I
                          I
                          I
                          I CUSTN
                    02    I S1
                   **************
                   *C_MAILCODE  **
                   *C_LST_NM    **
                   *C_FRST_NM   **
                   *C_PHONE_NO  **
                   *            **
                   **************
                    **************
                          I
                          I
                          I
                          I MINRPROP
                    03    I S1
                   **************
                   *TAG_NO      **I
                   *PUR_REQ_NO  **I
                   *FSC_CODE    **I
                   *MANF_SER_NO **I
                   *            **
                   **************
                    **************
                          I
                          I
                          I
                          I FSCSEG
                    04    I KU
                   ..............
                   :FSC_CODE    :K
                   :FSC_FIL     :
                   :FSC_DESC    :
                   :            :
                   :            :
                   :............:
                             FSCFILE
```

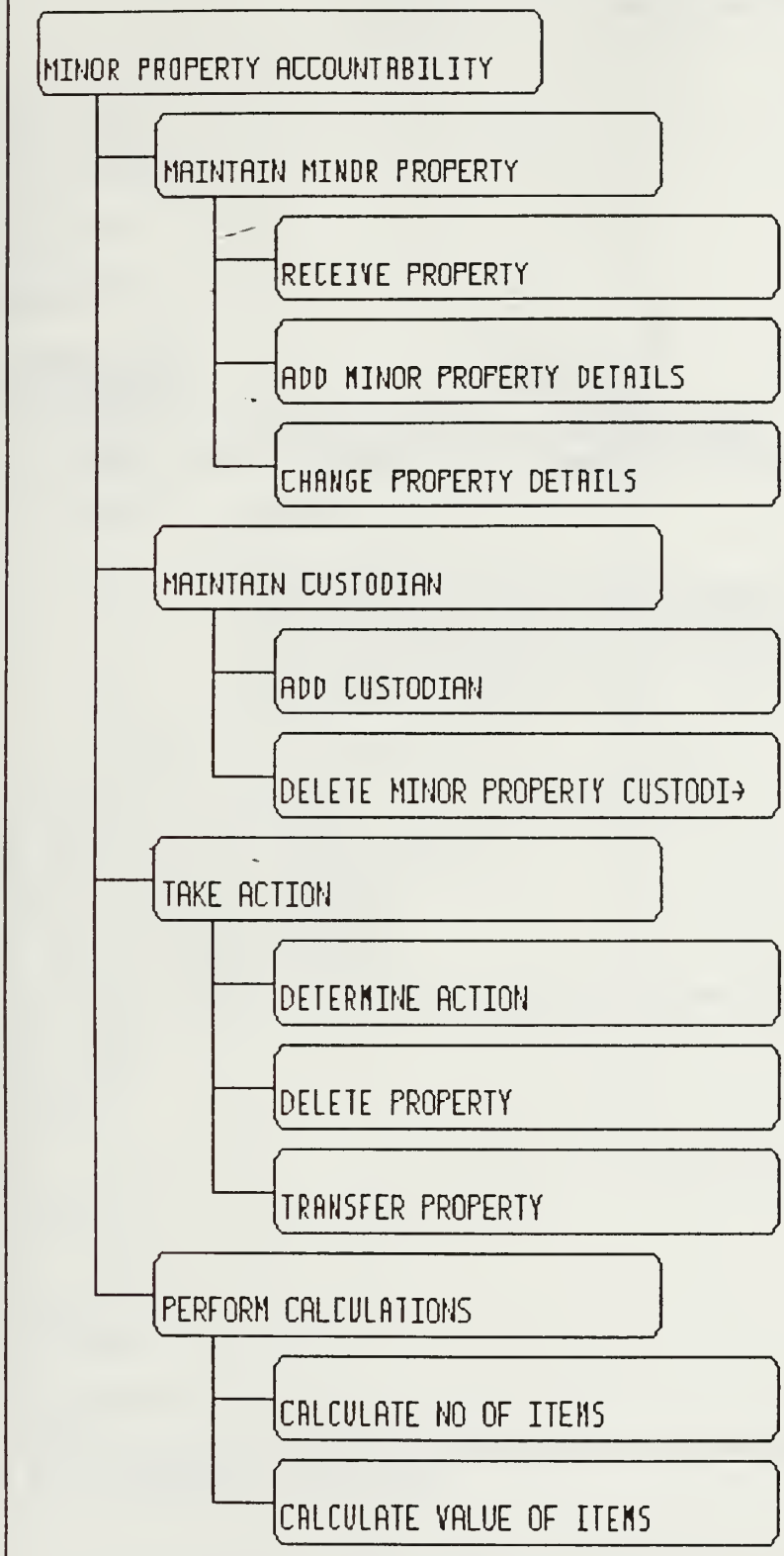
Master File Description Diagram for the MPA System (FOCUS)

Model : MINOR PROPERTY ACCOUNTABILITY Sept. 09, 1992 12:24
Subset: (complete model) page 1

Activity Hierarchy

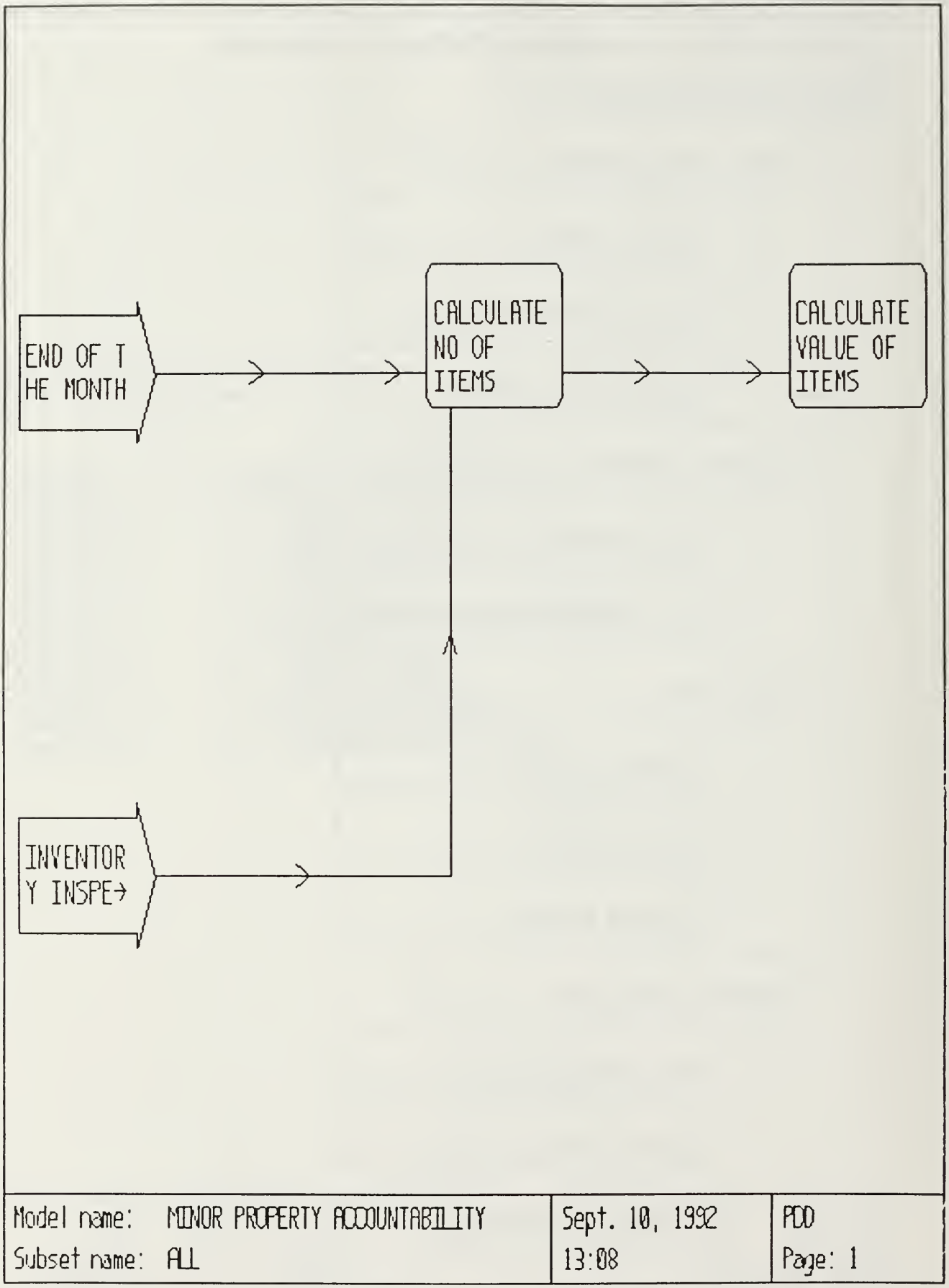
Function	1	MINOR_PROPERTY_ACCOUNTABILITY
Function	2	1 MAINTAIN_MINOR_PROPERTY
Elem Proc	3	1.1 RECEIVE_PROPERTY
Elem Proc	3	1.2 ADD_MINOR_PROPERTY_DETAILS
Elem Proc	3	1.3 CHANGE_PROPERTY_DETAILS
Function	2	2 MAINTAIN_CUSTODIAN
Elem Proc	3	2.1 ADD_CUSTODIAN
Elem Proc	3	2.2 DELETE_MINOR_PROPERTY_CUSTODIAN
Function	2	3 TAKE_ACTION
Elem Proc	3	3.1 DETERMINE_ACTION
Elem Proc	3	3.2 DELETE_PROPERTY
Elem Proc	3	3.3 TRANSFER_PROPERTY
Function	2	4 PERFORM_CALCULATIONS
Elem Proc	3	4.1 CALCULATE_NO_OF_ITEMS
Elem Proc	3	4.2 CALCULATE_VALUE_OF_ITEMS

Activity Hierarchy Report (IEF)



Model name: MINOR PROPERTY ACCOUNTABILITY	Sept. 10, 1992	AHD
Subset name: ALL	13:05	Page: 1

Activity Hierarchy Diagram for the MPA System (IEF)



Process Dependency Diagram for Perform Calculations (IEF)

```
-SET &OPT=' ';
-SET &OPTR='  ';
-SET &&PFKEY='    ';
-SET &DEPT='MR';
-TOPMENU
-SET &MSG=' ';
-RETRY
-SET &OPT=' ';
-CRTFORM 1
-" "
-" MPA(MIS/JLH) "
-"<D.&DATE/08"
-" ************************************************************************"
-" *               MINOR PROPERTY MAINTENANCE MENU                        *"
-" ************************************************************************"
-" "
-"    CODE         OPTION"
-"  "
-"     A           Add a new Property Item"
-"     C           Change data in an existing Property Item"
-"     L           List Manufacture Serial Number by Key Description"
-"     M           Modify DEPT/CUSTN/TAG Keys - Instructions"
-"     F           FSC Code and Nomenclature Lookup"
-"     R           Reporting Property Items"
-" "
-"    <T.&OPT>  <+5  ENTER CODE OR 'X' TO EXIT"
-"</9"
-IF &&PFKEY EQ 'PF03' OR 'PF15' THEN GOTO EXIT;
-IF &OPT EQ X GOTO EXIT;
-IF &OPT EQ A OR C GOTO GOODOPT;
-IF &OPT EQ L GOTO GOODSER;
-IF &OPT EQ M GOTO GOODMSG;
-IF &OPT EQ F GOTO RUNFSC;
-IF &OPT EQ R GOTO RUNRPTS;
-SET &MSG='Please enter a valid response - Press ENTER to Continue...';
-TYPE &MSG
-READ SYSIN
-GOTO RETRY
-RUNFSC
EX FSCRUN
-RUN
-GOTO TOPMENU
-GOODOPT
EX MODMP&OPT
-RUN
-GOTO TOPMENU
-GOODSER
EX MPASER
-RUN
-GOTO TOPMENU
-GOODMSG
-* This explains process for modifying key values and executes the MPA
-* change module if requested.
-CRTFORM LINE 3
-" "
```

FOCEXEC for the MPA System Main Menu (FOCUS)

```
-" ************************************************************************"
-" *              MINOR PROPERTY MODIFYING KEY VALUES                     *"
-" ************************************************************************"
-" "
-"To modify key values, enter appropriate code for excess property field"
-"(use T for Trnasfering an item or changing of keyed entry error); then"
-"notify Minor Property of the transaction by submitting a property "
-"transfer form or an excess property memo.  The Minor Property"
-"Accountability Officer will make the modifications to the data base."
-" "
-" "
-"Press ENTER to Continue..."
-"</9"
-GOTO TOPMENU


-RUNRPTS
-RPTRETRY
-SET &OPT=' ';
-SET &OPTR='  ';
-CRTFORM 1
-" "
-" MPA(MIS/JLH) "
-"<D.&DATE/08"
-" ************************************************************************"
-" *              MINOR PROPERTY MAINTENANCE MENU                         *"
" ************************************************************************"
-" "
-"    CODE        OPTION"



-"  "
-"     R1         Sum Items and Cost; Total and Department"
-"     R2         Tag Report by Department or Tag Item"
-"     R3         Tag Report by Department (11 X 8 1/2)"
-"     R4         Rejected Records Report"
-"     R5         Duplicated Records Report"
-"     R6         FSC Code Report by Department"
-"     R7         FSC Code and Nomenclature Listing"
-"     R8         Manufacture Serial Number Report"
-" "
-"    <T.&OPTR>  <+5  ENTER CODE OR 'X' TO EXIT"
-"</9"
-IF &&PFKEY EQ 'PF03' OR 'PF15' THEN GOTO EXIT;
-IF &OPTR EQ X GOTO EXIT;
-IF &OPTR EQ R1 OR R2 OR R3 OR R4 OR R5 OR R6 OR R7 OR R8
- THEN GOTO GOOD |¦ &OPTR;
-SET &MSG='PLEASE ENTER A VALID RESPONSE';
-TYPE &MSG
-SYSIN
-GOTO RPTRETRY
-GOODR1
EX MPATOT
-RUN
-GOTO RPTCHK
-GOODR2
EX MINTABZ
-RUN
-GOTO RPTCHK
-GOODR3
EX MINLAN
-RUN
-GOTO RPTCHK
-GOODR4
```

FOCEXEC for the MPA System Main Menu (continued) (FOCUS)

```
EX MPAREJS
-RUN
-GOTO RPTCHK
-GOODR5
EX MPADUPS
-RUN
-GOTO RPTCHK
-GOODR6
EX MINTABFS
-RUN
-GOTO RPTCHK
-GOODR7
EX FSCTAB
-RUN
-GOTO RPTCHK
-GOODR8
EX MPASER
-RUN
-GOTO RPTCH
-RPTCHK
-TYPE "Press ENTER to Continue..."
-READ SYSIN
-CRTFORM
-" "
-" "
-"</5 Report Completed...."
-"    Enter 'C' to Continue, 'R' for Reports, or 'X' to Exit:  <T.&OPT>"
-"</9"
-IF &&PFKEY EQ 'PF03' OR 'PF15' THEN GOTO EXIT;
-IF &OPT EQ X GOTO EXIT;
-IF &OPT EQ R GOTO RPTRETRY;
-GOTO TOPMENU
-EXIT
```

FOCEXEC for the MPA System Main Menu (continued) (FOCUS)

```
MINOR_PROPERTY_MAINTENANCE
  IMPORTS:
    Entity View import_new custodian (Optional)
      cust_code (Optional)
    Entity View import custodian (Mandatory)
      cust_code (Mandatory)
    Entity View import fsc (Mandatory)
      fsc (Mandatory)
    Entity View hidden_import property (Optional)
      tag_number (Mandatory)
    Entity View import property (Mandatory)
      tag_number (Mandatory)
      original_cost (Mandatory)
      current_cost (Optional)
      sn (Optional)
      model_version (Optional)
      year_acquired (Optional)
      inventory_date (Mandatory)
      bldg (Mandatory)
      room (Mandatory)
      mfr (Mandatory)
      source_document (Mandatory)
      po (Mandatory)
      property_pass_number (Optional)
      remarks (Optional)
      action (Optional)
  EXPORTS:
    Entity View export_new custodian
      cust_code
      dept_code
    Entity View export custodian
      cust_code
      dept_code
      lastname
    Entity View export fsc
      fsc
      description
    Entity View hidden_export property
      tag_number
    Entity View export property
      tag_number
      original_cost
      current_cost
      sn
      model_version
      year_acquired
      inventory_date
      bldg
      room
      mfr
      source_document
      po
      property_pass_number
      remarks
      action
```

Procedure Action Diagram for Minor Property Maintenance (IEF)

```
  LOCALS:
    Entity View work custodian
      cust_code
    Entity View work property
      tag_number
      original_cost
      current_cost
      sn
      model_version
      year_acquired
      inventory_date
      bldg
      room
      mfr
      source_document
      po
      property_pass_number
      remarks
      action
  ENTITY ACTIONS:

EXIT STATE IS all_ok
MOVE import property  TO export property
MOVE import fsc  TO export fsc
MOVE import custodian  TO export custodian
MOVE import_new custodian  TO export_new custodian

┌─ CASE OF COMMAND
├─ CASE add
  USE add_minor_property
      WHICH IMPORTS: Entity View import fsc
                     Entity View import custodian
                     Entity View import property
      WHICH EXPORTS: Entity View export fsc
                     Entity View export custodian
                     Entity View work property
   ┌─ IF EXITSTATE IS EQUAL TO all_ok
   │  MOVE work property  TO export property
   │  MOVE import fsc  TO export fsc
   │  MOVE import custodian  TO export custodian
   │  MOVE work property  TO hidden_export property
   └──
├─ CASE transfer
   ┌─ IF hidden_import property tag_number IS EQUAL TO import
   │              property tag_number
   │  USE transfer_minor_property
   │      WHICH IMPORTS: Entity View import_new custodian
   │                     Entity View import property
   │      WHICH EXPORTS: Entity View work custodian
   │                     Entity View work property
   │    ┌─ IF EXITSTATE IS EQUAL TO all_ok
   │    │  SET export_new custodian cust_code TO SPACES
   │    │  MOVE work property  TO export property
   │    │  MOVE work custodian  TO export custodian
   │    │  MOVE work property  TO hidden_export property
   │    └──
   ├─ ELSE
   │  EXIT STATE IS display_before_update
   └──
```

Procedure Action Diagram for Minor Property Maintenance (continued) (IEF)

```
CASE change
  IF import property tag_number IS EQUAL TO hidden_import
                property tag_number
  USE change_minor_property_details
      WHICH IMPORTS: Entity View import property
      WHICH EXPORTS: Entity View work property
    IF EXITSTATE IS EQUAL TO all_ok
    MOVE work property  TO export property
    MOVE work property  TO hidden_export property
  ELSE
  EXIT STATE IS display_before_update
CASE delete
  IF import property tag_number IS EQUAL TO hidden_import
                property tag_number
  USE delete_minor_property
      WHICH IMPORTS: Entity View import property
      WHICH EXPORTS: Entity View work property
  ELSE
  EXIT STATE IS display_before_update
CASE display
  USE display_minor_property
      WHICH IMPORTS: Entity View import property
      WHICH EXPORTS: Entity View work property
                     Entity View export fsc
                     Entity View export custodian
  IF EXITSTATE IS EQUAL TO all_ok
  MOVE work property  TO export property
  MOVE import fsc  TO export fsc
  MOVE import custodian  TO export custodian
  MOVE work property  TO hidden_export property
OTHERWISE
EXIT STATE IS invalid_command
```

Procedure Action Diagram for Minor Property Maintenance (continued) (IEF)

```
MODIFY FILE MINOR
-* THIS INCLUDES DATA FOR MINOR PROPERTY DATABASE
COMPUTE PFKEY/A4=;
COMPUTE RCODE/I1=;
COMPUTE XFSCNM/A60=;
CRTFORM LINE 1
"*************************************************************<0X
                                                              "
           MINOR PROPERTY CONTROL SYSTEM"
"*************************************************************<0X
" "
"Create a new minor property record by entering the following data:"
" "
"      Department Code                                  <0X
<DEPT_CODE/05>"
"      Custodian Mail Code                              <0X
<C_MAILCODE/04>"
"      TAG Number                                       <0X
<TAGNO/07>"
"      FSC Code                                         <FSCNO/04>"
" "
"*************************************************************<0X
"             Please press the ENTER key to Continue"
"              (or hit PF2 to Cancel / PF3 to Quit)"
"*************************************************************<0X
IF PFKEY EQ 'PF02' GOTO TOP
  ELSE IF PFKEY EQ 'PF03' GOTO EXIT;
COMPUTE RCODE = LOOKUP(FSC_DESC);
MATCH DEPT_CODE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH C_MAILCODE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH TAG_NO
ON NOMATCH TYPE
"DEPT CODE  <DEPT_CODE       MAIL CODE  <C_MAILCODE       TAG NO  <0X
<TAGNO     FSC CODE  <FSCNO  "
"FSC DESC   <FSC_DESC "
ON NOMATCH CRTFORM LINE 1
" "
"* * * * * * * * * *      ADDING A MINOR PROPERTY ITEM      <0X
* * * * * * * * * *"
" "
"Cust Last Name    <CUSTNLN/10>       Cust First Name      <0X
<CUSTNFN/10>"
"Cust Phone No     <CPHONE/07>"
" "
"Source Doc No     <DOCNO/10>       Purchase Request No  <0X
<PURREQ/07>"
"Year Acquired     <ACQDT/02>               Initial Cost         <0X
<ACQCOST/06>"
"Manf Model No     <MODNO/12>     Manf Serial No       <0X
<SERNO/16>"
"Manf Name         <MFGNM/25>"
" "
"Item Desc         <IDESC/35>"
"Building Number   <IBLDG/04>             Room Number          <0X
<IROOM/04>"
"Inventory YMD     <INVDT/06>           Adusted Cost         <0X
<ADJCOST/06>"
"Inventory Remarks <INVRMKS/22>"
" "
"AIS Code          <AISCD/10>       Subsystem Code       <0X
<SUBCD/10>"
```

FOCEXEC for Changing a Minor Property Item (FOCUS)

```
" "
"*************************************************************<0X
"   Press the ENTER to Add this Record (or hit PF2 to Cancel<0X
/ PF3 to Quit)"                                                    ON
NOMATCH TYPE                                                       "
...RECORD HAS BEEN ADDED. "                                        ON
NOMATCH INCLUDE                                                    ON
MATCH GOTO CASEX                                                 CASE
CASEX                                                           MATCH
TAG_NO                                                        ON MATCH
CRTFORM
"*************************************************************************"
"                  RECORD IS ALREADY IN THE DATA BASE"
"*************************************************************************"
ON MATCH REJECT
ENDCASE
DATA
CHECK 1
END
-EXIT
```

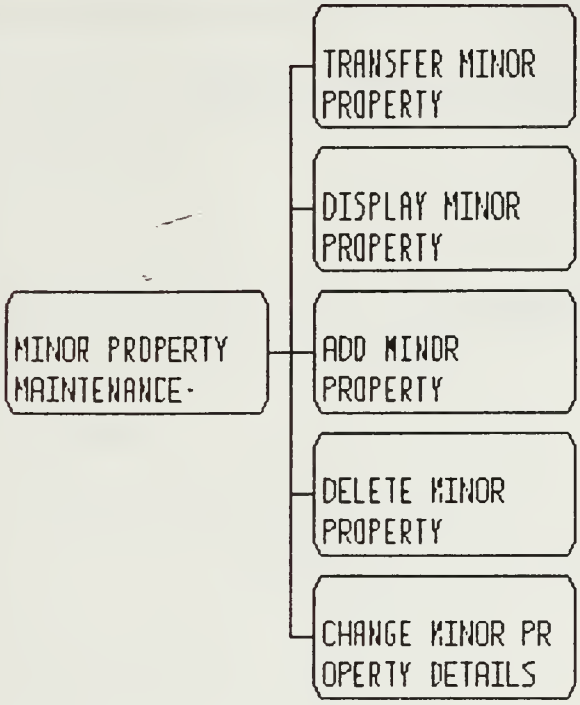
BSD Action Block: CHANGE_MINOR_PROPERTY_DETAILS Sept. 09, 1992 13:10

```
CHANGE_MINOR_PROPERTY_DETAILS
  IMPORTS:
    Entity View import property (Mandatory)
      tag_number (Mandatory)
      original_cost (Mandatory)
      current_cost (Optional)
      sn (Optional)
      model_version (Optional)
      year_acquired (Optional)
      inventory_date (Mandatory)
      bldg (Mandatory)
      room (Mandatory)
      mfr (Mandatory)
      source_document (Mandatory)
      po (Mandatory)
      property_pass_number (Optional)
      remarks (Optional)
      action (Optional)
  EXPORTS:
    Entity View export property
      tag_number
      original_cost
      current_cost
      sn
      model_version
      year_acquired
      inventory_date
      bldg
      room
      mfr
      source_document
      po
      property_pass_number
      remarks
      action
  LOCALS:
  ENTITY ACTIONS:
    Entity View property
      tag_number
      original_cost
      current_cost
      sn
      model_version
      year_acquired
      inventory_date
      bldg
      room
      mfr
      source_document
      po
      property_pass_number
      remarks
      action

  READ property
        WHERE DESIRED property tag_number IS EQUAL TO import
              property tag_number
  WHEN successful
```

Process Action Diagram for Changing a Minor Property Item (IEF)

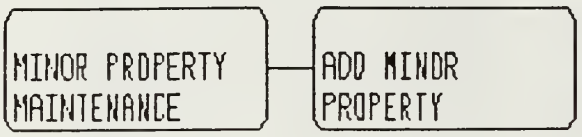
```
UPDATE property
SET original_cost TO import property original_cost
SET current_cost TO import property current_cost
SET sn TO import property sn
SET model_version TO import property model_version
SET year_acquired TO import property year_acquired
SET inventory_date TO import property inventory_date
SET bldg TO import property bldg
SET room TO import property room
SET mfr TO import property mfr
SET source_document TO import property source_document
SET po TO import property po
SET property_pass_number TO import property
          property_pass_number
SET remarks TO import property remarks
SET action TO import property action
WHEN successful
MOVE property  TO export property
WHEN not unique
EXIT STATE IS property_nu
WHEN not found
EXIT STATE IS property_nf
```

Process Action Diagram for Changing a Minor Property Item (continued) (IEF)



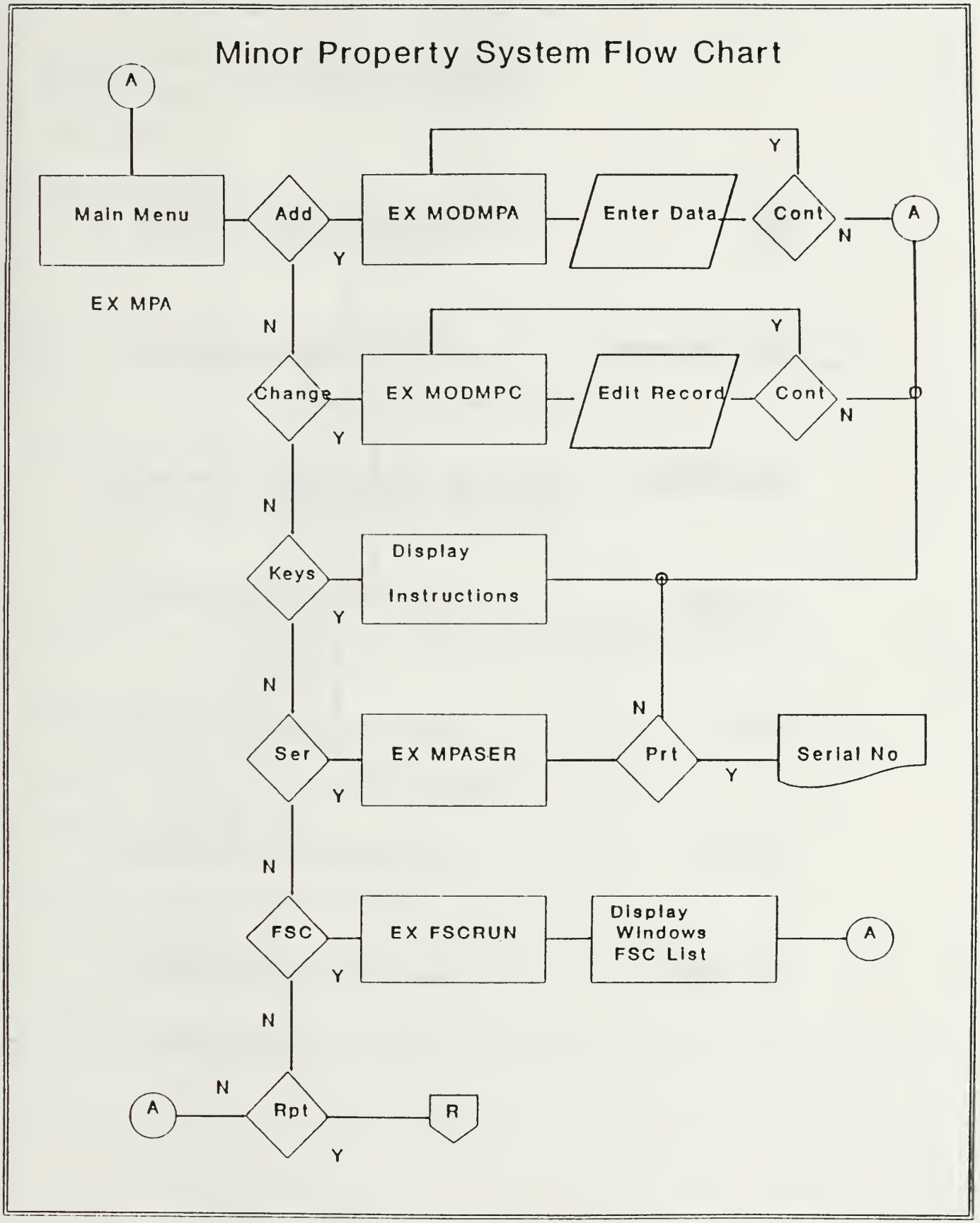
Model name: MINOR PROPERTY ACCOUNTABILITY	Sept. 10, 1992	MTX
Subset name: ALL	13:23	Page: 1

Structure Chart Diagram for the MPA System (IEF)

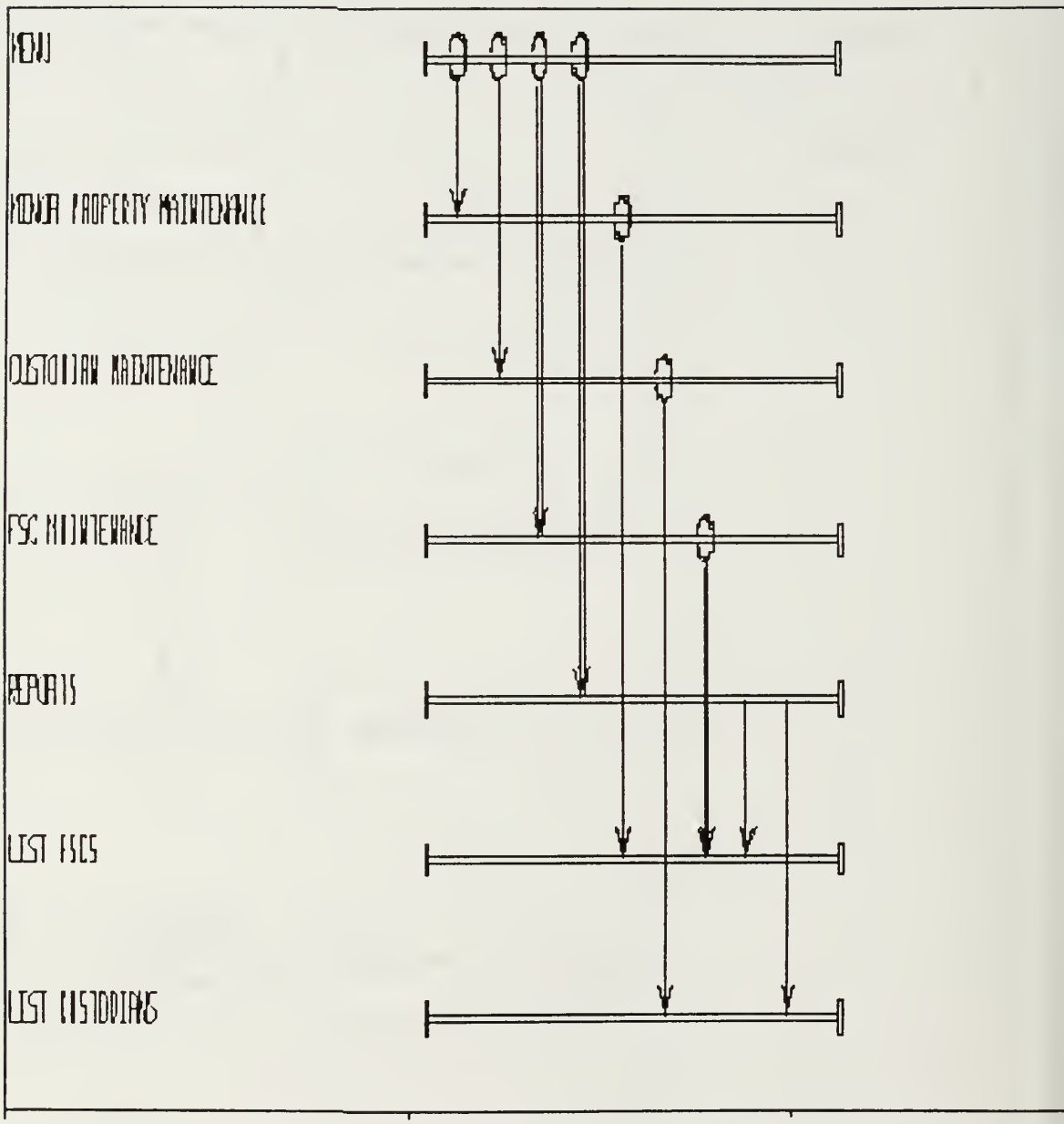


Model name:	MINOR PROPERTY ACCOUNTABILITY	Sept. 10, 1992	MTX
Subset name:	ALL	13:28	Page: 1

Action Block Usage Diagram for Add Minor Property (IEF)



MPA System Flow Chart



Dialog Flow Chart for the MPA System (IEF)

Procedure Step Definition

Business System MINOR_PROPERTY_ACCOUNTABILITY
Procedure MINOR_PROPERTY_MAINTENANCE
Procedure Step MINOR_PROPERTY_MAINTENANCE

Description:

PF Key Definitions:

Key	Command	Type	Screen Display
06	FSC	Local	YES

Procedure Step Definition

Dialog Flow Link from
Business System MINOR_PROPERTY_ACCOUNTABILITY
Procedure MINOR_PROPERTY_MAINTENANCE
Procedure Step MINOR_PROPERTY_MAINTENANCE

To
Procedure LIST_FSCS
Procedure Step LIST_FSCS

Description: FLOW TO LIST FSC WHEN THE USER ENTERS THE WRONG FSC, CAN SELECT FROM THE FSC LIST AND WILL RETURN THE SELECTED FSC

Flows on exit states:
FSCS_NF
Autoflow on LIST command.

Execute destination with DISPLAY command.

Returns on exit states:
RETURN_REQUESTED
Autoflow on RETURN command.

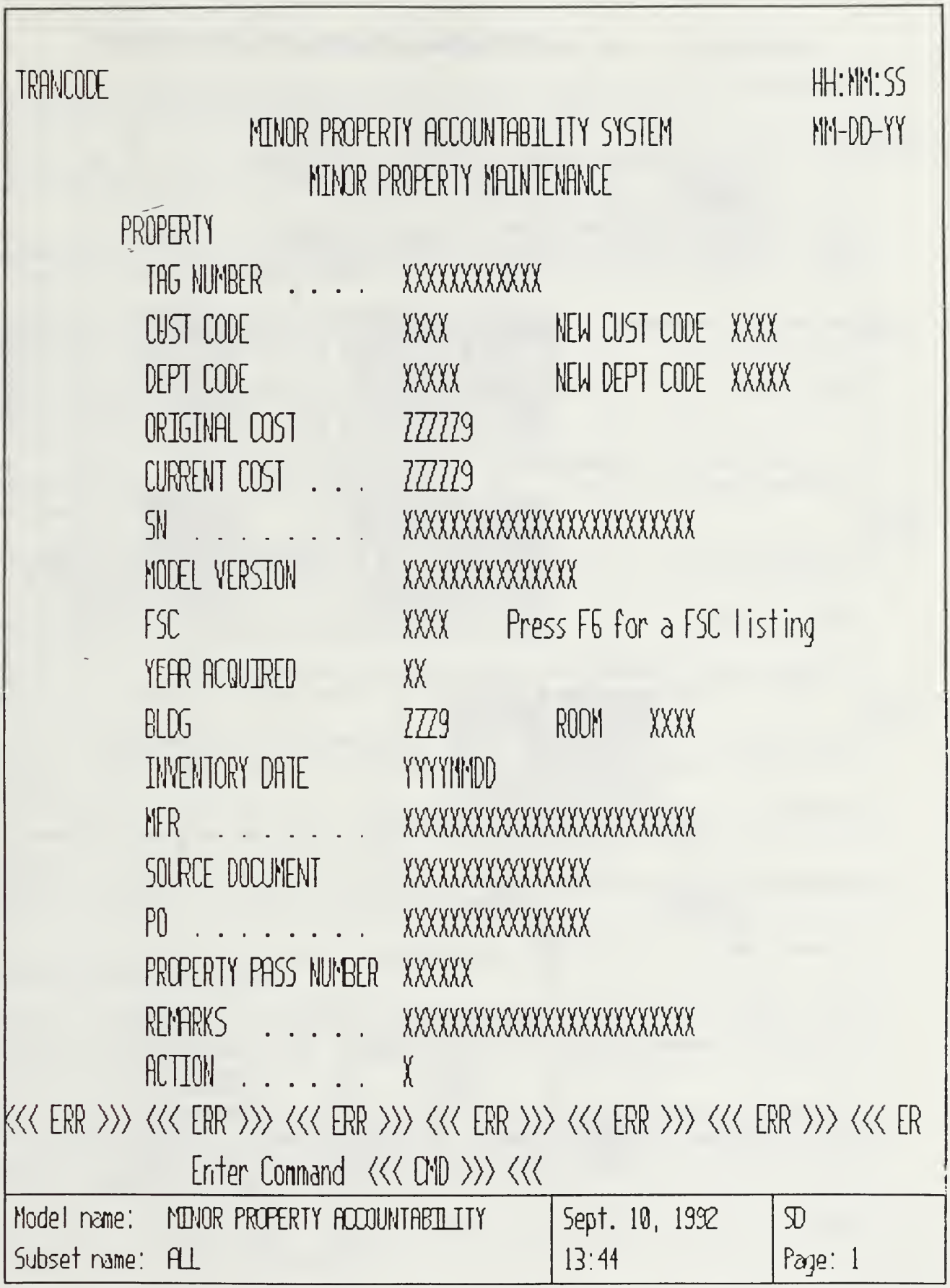
Display on return with no command.

Data returned to
View IMPORT of entity FSC
Attributes:
FSC
from
View RETURN_FROM_LINK of entity FSC
Attributes:
FSC

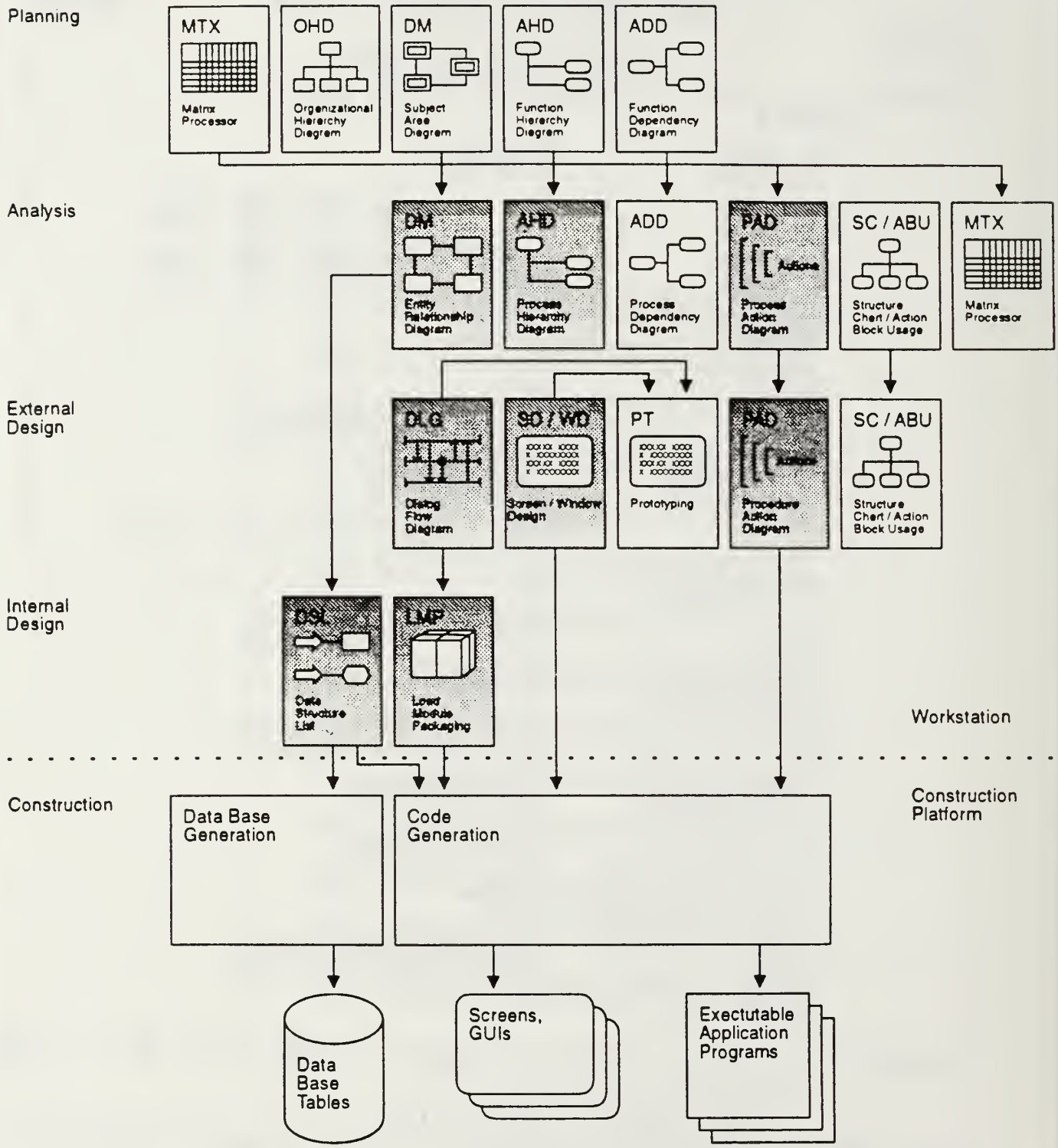
Procedure Step Definition Report from Minor Property Maintenance to List FSC's (IEF)

```
                                                          HH:MM:SS
              MINOR PROPERTY ACCOUNTABILITY SYSTEM        MM-DD-YY
                          LIST FSCS

          SEL  FSC  DESCRIPTION
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           X   XXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

                            LINES XXX TO XXX OF XXX
<<< ERR >>> <<< ERR >>> <<< ERR >>> <<< ERR >>> <<< ERR >>> <<< ERR >>> <<< ER
```

List FSC Screen for the MPA System (IEF)



Minor Property Maintenance Screen (IEF)



IEF Support for Information Engineering
(Shaded Diagrams Required for Construction)

LIST OF REFERENCES

Alavi, M., "Mixing Prototyping and Data Modeling for Information-System Design," **IEEE Software**, pp. 86-91, May 1991.

Agresti, W. W., **New Paradigms for Software Development**, pp. 1-11, IEEE Computer Society Press, 1986.

Bloor, R., "Repository Technology: CASE tools that use repository technology will provide significant increase in development," **DBMS**, v.4, pp. 17-19, December 1991.

Burden, L., "Rapid Application Development," **Computer Conference Newsletter**, pp. 6-7, 7 May 1991.

Burke, J. P., "Tough CASE," **HP Professional**, v.5, pp. 30-37, July 1991.

Case, A. F., **Information Systems Development: Principles of Computer-Aided Software Engineering**, pp. 150-160, Prentice-Hall, 1986.

Chikofsky, E. J., Martin, D. E., and Chang, H., "Assessing the STate of Tools Assessment," **IEEE Software**, pp. 18-21, May 1992.

Computerworld, "CASE Tools pass benchmark," p. 60, 2 March 1992.

Corbin, D. S., "Establishing the Software Development Environment," **Journal of Systems Management**, v.42, p. 28-32, September 1991.

Datapro, **Information Engineering Facility**, pp. 1-7, July 1991; reprint, McGraw-Hill Inc. (page references are to reprint edition)

Due, R. T., "In pursuit of enterprise maodeling," **Database Progamming and Design**, v.4, pp. 54-59, September 1991.

Eastwood, Alison, "In focus: RAD: not an instant fix," **Computing Canada**, v.17, pp. 17-19, 15 August 1991.

Eliot, L., "Information Engineering Facility," **CASE Trends**, November/December 1991.

National Bureau of Standards Publication 500-148, **Application Software Prototyping and Fourth Generation Languages**, by G. E. Fisher, 1987.

Floyd, C., "A Comparative Evaluation of System Development Methods, " in **Information Systems Design Mehodologies: improving the practice**, Olle, T.E, Sol, A. A., Verrijin-Stuart, ed. pp. 19-37, North-Holland, 5-7 May, 1986.

Forte, G., "Tools Fair: Out of the Lab, Onto the Shelf," **IEEE Software**, pp. 70-75, May 1992.

Frey, W., **Computer-Aided Software Engineering (CASE) Environment Issues**, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1987.

Ginsberg M. and others, "Issues Involved in Sofware Methods Selection and Evaluation, " in **Second International Workshop on Computer-Aided Software Engineering Advance Working Papers Volume 2, CASE '88, Cambridge, Massachusetts, July 12-15, 1988**, Chikofsky, E. ed., pp. 19-7 -- 19-10, 1988.

Guvarin, S.L., "Where does Prototyping Fit in IS Development," **Journal of Systems Management**, v. 42, pp. 13-16, February 1991.

Haas, M. S. and Hochstetler, M. L., **Information Enginering of the Curricular Officers' Segment of a Unified STudent Academic Databse System for NPS**, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

Interviews between J. Harr, Programmer/Analyst, Naval Postgraduate School, Monterey, California, and the author, 18, 20, 21, and 25 August.

Information Builders, Inc., "James Martin," **FOCUS News**, pp. 35-40, Summer/Fall 1990.

Information Builders, Inc., **Support Services for your Information Center**, pp. 3-16.

Information Builders, Inc., **PC/FOCUS Release 6.0 for DOS Product Fact Sheet**, 1991.

Information Builders, Inc., **FOCUS for IBM Mainframe, Users Manual Release 6.5, Vol. I**, 1990.

Information Builders, Inc., **FOCUS for IBM Mainframe, Users Manual Release 6.5, Vol. II**, 1990.

Jaakkola, J. E., and Drake, K. B., "ASDM: The Universal Systems Development Methodology," **Journal of Systems Management**, v.42, pp. 6-11, February 1991.

James Martin & Co., "Architecting Enterprises for the 21st Century, brochure, 1992.

Johnston, M. W., "European software umbrella," **Datamation**, v. 37, pp. 32-34, 1 March 1991.

Kemerer, C. F., "Research Problems in the Managerial Evaluation of Computer-Aided Software Engineering Tool Impacts," in **Second International Workshop on Computer-Aided Software Engineering Advance Working Papers Volume 2, CASE '88, Cambridge, Massachusetts, July 12-15, 1988**, Chikofsky, E. ed., pp. 17-3 - 17-6, 1988.

Keuffel, W., "Faking Top-Down Development," **Computer Language**, v.8, pp. 35-40, September 1991.

Kemerer, C. F., "How the Learning Curve Affects CASE Tool Adoption," **IEEE Software**, pp. 23-28, May 1992.

Keys, J., "How sofware is devleoped undergoing basic changes; with GUI's, servers, objects and parallellism, **Software Magazine**, v. 12, pp. 38-47, January 1992.

Loh, M., and Weston, R., "Reaping CASE Harvests," **Datamation**, pp. 31-34, 1 July 1989.

Manley, G., **Classification and Evaluation of CASE Tools**, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1990.

Martin, J., **Rapid Application Development**, MacMillian Publishing Company, 1991.

Martin, J., **Information Engineering (Book I: Introduction)**, Prentice-Hall, 1989.

Martin, J., **Information Engineering (Book II: Planning and Analysis)**, Prentice-Hall, 1990.

Martin, J., **Information Engineering (Book III: Design and Contruction)**, 1990.

Martin, J., **Fourth Generation Languages, Volume I, Principles**, Prentice-Hall, 1985.

Martin, J., **Fourth Generation Languages, Volume II, Representative 4GLS**, Prentice-Hall, pp.139-185, 1986.

Telephone conversation between J. McGrail, Team Leader, CASE Technology, US Army Materiel Command Systems Integration and Management Activity, and the author, June 1992.

McIninch, D., "Achieving Your Return on Investment in CASE," **Database Management**, pp. 16-17, April 1992.

Page-Jones, M., "The CASE Manifesto," **CASE Outlook**, January-February 1992.

Paul, L. G., "Information Builders Inc. (The Datamation 100) (Company Profile)," **Datamation**, v.38, pp. 64-65, 15 June 1992.

McParland, P., "Playing the generation game," **EXE**, v.6, p. 14-18, June 1991.

Plauger, P.J., "Heresies of software design," **Computer Language**, v.8, February 1991.

Pressman, R. S., **Software Engineering, 3rd edition: A Practioner's Approach**, pp. 22-38, McGraw-Hill, Inc., 1992.

Interviews between M. Spencer, Director, Management Information Systems, Naval Postgraduate School, Monterey, California, and the author, 6 June 1992, 27 June 1992.

Software Technology Support Center, **CrossTalk**, p. 1., March 1992.

Sullivan-Trainor, M. L., "TI's IEF scores high for integration, benefits delivery," **Computerworld**, pp. 72-77, 22 April 1991.

Swartout, W. and Blazer, R., "On the Inevitable Intertwining of Specification and Implemetnation," in **New Paradigms for Software Development**, ed. W. Agresti, pp. 26-29, 1986.

Teledyne Brown Engineering, Technical Report MC89-S/W-METH-0001, **An Approach to Evaluating Software Methods** by R. Pirchner and others, March 1989.

Texas Instruments, **Business Area Analysis I, Student Guide Release 4.11**, 1992.

Texas Instruments, "Introducing IEF 5.0," product brochure, 1991.

Texas Instruments, **A Guide to Information Engineering Using the IEF, 2nd edition**, Texas Instruments, Inc., 1990.

Texas Instruments, **Rapid Development Using the IEF**, Texas Instruments, Inc., 11 January 1991.

Texas Instruments, **IEF Development Tutorial**, Texas Instruments, Inc., 11 February 1991.

Texas Instruments, **Education Schedule 1992.**

Uluakar, T., **From Stuructured Methods to Information Engineering, A Comparison**, Texas Instruments Inc., pp. 1- 13, March 1991.

Whitten, J.L., Bentley, L. D., and Marolow, V. M., **System Analysis and Design Methods, 2nd edition**, pp. 110-129, Irwin, 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dean Barry A. Frew, Code 05 Dean of Computer and Information Services Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Myung Suh, Code AS/Su Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
5.	Ms. Lucille C. Clark, Code 53 Management Information Systems Naval Postgraduate School Monterey, California 93943-5000	1
6.	Mr. Michael P. Spencer, Code 53 Management Information Systems Naval Postgraduate School Monterey, California 93943-5000	1
7.	Ms. Jeffrie Penrod Texas Instruments Info Tech Group/IEF Group 5353 Betsy Ross Drive Santa Clara, California 95054	1
8.	Professor Tung Bui, Code AS/Bd Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
9.	Mr. John P. McGrail U.S. Army Material Command Systems Integration and Management Activity 1222 Spruce Street St. Louis, Missouri 63103-4159	1

10. Professor Richard Pirchner 1
Department of Computer Science
Monmouth College
West Long Branch, New Jersey 07764

11. Mr. John LeBaron 1
Software Engineering Directorate
U.S. Army Communications-Electronics Command
and Fort Monmouth
Fort Monmouth, New Jersey 07703-5000

12. Mr. Jim Van Buren 1
Software Technology Support Center
Hill Air Force Base, Utah 84056

13. Mr. Walter J. Utz, Jr. 1
Technology Transition Center
1132 Thorntree Court
San Jose, California 95120

Thesis
C48189 Clark
c.1 Information engineering and the Information Engineering Facility verus rapid application development and FOCUS.